DEFENSE TECHNICAL INFORMATION CENTER

Information for the Defense Community

DTIC® has determined on 05/24/2010 that this Technical Document has the Distribution Statement checked below. The current distribution for this document can be found in the DTIC® Technical Report Database.

☒ **DISTRIBUTION STATEMENT A.** Approved for public release; distribution is unlimited.

☐ **© COPYRIGHTED**; U.S. Government or Federal Rights License. All other rights and uses except those permitted by copyright law are reserved by the copyright owner.

☐ **DISTRIBUTION STATEMENT B.** Distribution authorized to U.S. Government agencies only (fill in reason) (date of determination). Other requests for this document shall be referred to (insert controlling DoD office)

☐ **DISTRIBUTION STATEMENT C.** Distribution authorized to U.S. Government Agencies and their contractors (fill in reason) (date of determination). Other requests for this document shall be referred to (insert controlling DoD office)

☐ **DISTRIBUTION STATEMENT D.** Distribution authorized to the Department of Defense and U.S. DoD contractors only (fill in reason) (date of determination). Other requests shall be referred to (insert controlling DoD office).

☐ **DISTRIBUTION STATEMENT E.** Distribution authorized to DoD Components only (fill in reason) (date of determination). Other requests shall be referred to (insert controlling DoD office).

☐ **DISTRIBUTION STATEMENT F.** Further dissemination only as directed by (inserting controlling DoD office) (date of determination) or higher DoD authority.

Distribution Statement F is also used when a document does not contain a distribution statement and no distribution statement can be determined.

☐ **DISTRIBUTION STATEMENT X.** Distribution authorized to U.S. Government Agencies and private individuals or enterprises eligible to obtain export-controlled technical data in accordance with DoDD 5230.25; (date of determination). DoD Controlling Office is (insert controlling DoD office).

Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation



May 9-11, 1995
Orlando, Florida
Sponsored by STRICOM·DMSO
Contract - N61339-92-C-0045

ALL BUT WAR IS SIMULATION

DMSO

INSTITUTE FOR SIMULATION & TRAINING

IST

IST-TR-95-04

INSTITUTE FOR SIMULATION AND TRAINING • TECHNICAL REPORT

Proceedings of The Fifth Conference on Computer Generated Forces and Behavioral Representation

May 9-11, 1995
Orlando, Florida

Sponsored by:
U.S. Army Simulation, Training, and Instrumentation Command
Defense Modeling and Simulation Office

Organized by:
Institute for Simulation and Training
3280 Progress Drive
Orlando, Florida 32826

University of Central Florida, Division of Sponsored Research

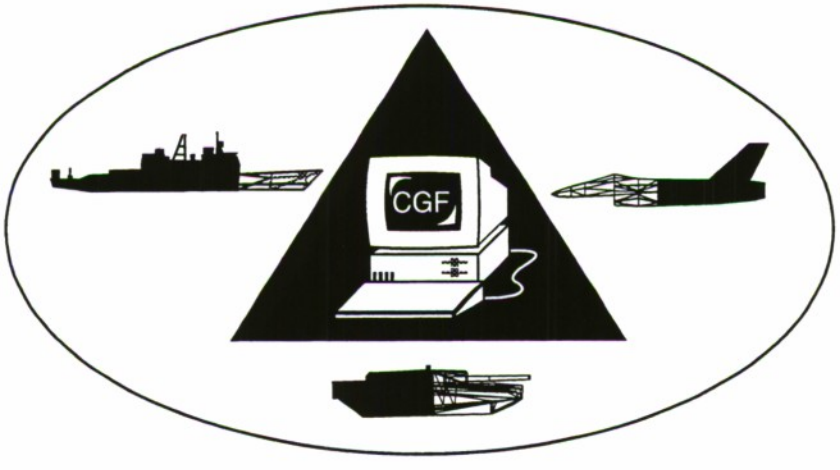
Contract N61339-92-C-0045 CDRL A00D
IST-TR-95-04

Reviewed By:

Daniel E. Mullally, Jr.

Michael Fineberg 703 578 2839

20100311117


CGF

Preface

PURPOSE

This report presents the proceedings of the Fifth Computer Generated Forces (CGF) and Behavioral Representations (BR) Conference. The Conference is scheduled from 9 to 11 May in Orlando, Florida and is hosted by the Institute for Simulation and Training (IST). IST is a component of the Division of Sponsored Research at the University of Central Florida.

OBJECTIVES

The objectives of this conference are to:

- Provide a forum for information exchange on CGF and BR modeling research.
- Identify gaps in CGF and BR research.
- Present upcoming research programs and opportunities.
- Determine the CGF and BR community interest in technology demonstrations.

Attendees will have an opportunity to participate in discussions of Service User needs, CGF systems issues, and technical presentations on the components of a CGF.

BACKGROUND

Under the sponsorship of the U.S. Army, Simulation, Training & Instrumentation Command (STRICOM) and Defense Modeling and Simulation Office (DMSO), the Institute for Simulation and Training, of the University of Central Florida is conducting this Fifth Conference on CGF and BR.

UCF/IST has hosted four previous CGF & BR symposia. An indication of the success of these interest group meetings is reflected in the steady increase in attendance, rising from 84 attendees in Oct. 1990 to 128 in May of 1991, to 310 in March of 1993, to 323 in May of 1994.

Following the topics outlined in the Second BR symposium, IST is tasked by STRICOM to host a continuing series of CGF and BR conferences. These conferences will provide a continuing ability to promote and focus research in this important area. Most attendees at previous conferences expressed an interest in continuing in a dialogue with training developers on future requirements in order to justify their own internal research and development participation and commitment to this emerging technology.

Other conference topics which merit consideration for resolution by the community of military, industry, and academic researchers in BR include:

- Interoperability Standards for Behavioral Representation in Defense Simulations:
- Validation, Verification and Accreditation of Behavioral Representation models:
- Functional Specification rationale for Behavioral Representation models in Design, Testing and Training Simulations;
- Interoperability issues for classified modeling in Behavioral Representation;
- Behavioral Representation in Virtual Reality.

GENERAL

This report is presented in one volume. Wherever possible, the papers are arranged in the order of presentation.

A list of attendees will be distributed to all registered attendees at the conclusion of the conference.

Conference Committee

Conference Chair

Daniel E. Mullally, Jr.

Program Committee

Douglas A. Reece

Clark R. Karr

Robert W. Franceschini

Production Assistance

Doug Barrett

Valerie Truhan

Vicki McGurk

Local Arrangements and Registration

Vince Amico

Linda Toth

Karen Gauvin

Deodith Mapas

Table of Contents

Preface i

Session 2a: Project Status Reports

ModSAF Development Status 3
Anthony J. Courtemanche, Andy Ceranowicz
Loral ADS
Cambridge, Massachusetts

The Distributed Interactive C3I Effectiveness (DICE) Simulation Project: An Overview 15
Mike Davies, Carsten Gabrisch
Information Technology Division, DSTO
Salisbury, S. Australia

Integrated Eagle/BDS-D: A Status Report 21
Robert W. Franceschini
UCF/IST
Orlando, Florida

Simulated Intelligent Forces for Air: The SOAR/IFOR Project 1995 27
John E. Laird, Randolph M. Jones, Frank Koss, Paul E. Nielsen, Michael van Lent, Robert E. Wray, III
Artificial Intelligence Lab, University of Michigan
Ann Arbor, Michigan
W. Lewis Johnson, Paul S. Rosenbloom, Karl Schwamb, Milind Tambe
Information Sciences Institute, USC
Marina del Rey, California
Jill F. Lehman, Robert Rubinoff, Julie Van Dyke
Computer Science Department, Carnegie Mellon University
Pittsburgh, Pennsylvania

Session 2b: Reasoning I

Building Intelligent Pilots for Simulated Rotary Wing Aircraft 39
Milind Tambe, Karl Schwamb, Paul S. Rosenbloom
ISI, USC
Marina del Rey, California

A Multiple Agent Hybrid Control Architecture for Automated Forces: Design & Software Implementation 45
Xiaolin Ge, John James, Anil Nerode
Mathematical Sciences Institute, Cornell University
Ithaca, New York

* **Context-based Representation of Intelligent Behavior in Simulated Opponents** 53
Avelino J. Gonzalez
Electrical & Computer Engineering Department, UCF
Orlando, Florida
Robert Ahlers
Naval Air Warfare Center, Training Systems Division
Orlando, Florida

Automated Agents That Learn and Explain Their Own Actions: A Progress Report 63
Sakir Kocabas, Ercan Oztemal, Mahmut Uldudag, Nazim Koc
Marmara Research Center
Gebze, Turkey

Session 3a: Constructive + Virtual Simulation

* **Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSD CLCGF** 71
Robert B. Calder, Jeffrey C. Peacock, Jr.
SAIC
Waltham, Massachusetts
James Panagos
TASC
Reading, Massachusetts
Thomas E. Johnson
Raytheon Company
Tewksbury, Massachusetts

Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF 83
Robert B. Calder, Jeffrey C. Peacock, Ben P. Wise
SAIC
Waltham, Massachusetts
Thomas Stanzione, Forrest Chamberlain, James Panagos
TASC
Reading, Massachusetts

Survey of Constructive + Virtual Linkages **93**
Matthew K. Kraus, David R. Stober, William F. Foss, Robert W. Franceschini, Mikel D. Petty
UCF/IST
Orlando, Florida

Disaggregation Overload and Spreading Disaggregation in Constructive + Virtual Linkages **103**
Mikel D. Petty, Robert W. Franceschini
UCF/IST
Orlando, Florida

Session 3b: Reasoning II

Natural Language Processing for IFORs: Comprehension and Generation in the Air Combat Domain **115**
Jill Fain Lehman, Julie Van Dyke, Robert Rubinoff
Carnegie Mellon University
Pittsburgh, Pennsylvania

Agent Tracking in Complex Multi-Agent Environments: New Results **125**
Milind Tambe, Paul S. Rosenbloom
ISI, USC
Marina del Rey, California

A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces **135**
Michael R. Hieb, Gheorge Tecuci, J. Mark Pullen
Department of Computer Science, George Mason University
Fairfax, Virginia
Andrew Ceranowicz
Loral ADS
Cambridge, Massachusetts
David Hille
ANSER
Arlington, VA

Session 4a: Command & Control Modeling I

An Automated CBS OPFOR **149**
Ian Page
Defence Research Agency
Kent, England
Gary Kendall
Logica UK Ltd.
London, England

Automated Mission Planning in ModSAF **159**
Clark R. Karr, Sumeet Rajput, Jaime E. Cisneros, Hai-Lin Nee
UCF/IST
Orlando, Florida

Multi-Application Command Agents **169**
Helen Lankester
Software Engineering Centre, Defence Research Agency
Kent, England

Session 4b: VV&A

Measuring Entity and Group Behaviors of Semi-Automated Forces **181**
Larry L. Meliza, Eric A. Vaden
U.S. Army Research Institute, Simulator Systems Research Unit
Orlando, Florida

The Use of Automated Regression and VVA Testing in ModSAF **193**
James Perneski, Paul Monday
Loral ADS
Cambridge, Massachusetts

Verification and Validation of Modular Semi-Automated Forces (ModSAF) in Support of A2ATD Experiment 1 ..... **197**
John G. Thomas
U.S. Army Materiel Systems Analysis Activity
Aberdeen Proving Ground, Maryland

Session 5a: Command & Control Modeling II

Command Entity Cognitive Behaviors for SAF and CGF **203**
Howard Mall, Kent Bimson, Jenifer McCormack, Dirk Ourston
SAIC
Orlando, Florida

✳ **Intelligent Computer Generated Forces for Command and Control** **211**
Paul E. Nielsen
Department of Electrical Engineering and Computer Science, University of Michigan
Ann Arbor, Michigan

Autonomous Agent Interactions in ModSAF **219**
David R. Pratt, Gary McAndrews, Robert McGhee
Department of Computer Science - Naval Postgraduate School
Monterey, California

Session 5b: Route Planning I

Route Planning in CCTT 233
Chuck Campbell, Richard Hull, Eric Root, Lance Jackson
SAIC
Orlando, Florida

Dynamic Obstable Avoidance for Computer Generated Forces 245
Clark R. Karr, Michael A. Craft, Jaime E. Cisneros
UCF/IST
Orlando, Florida

Path Planning With Terrain Utilization in ModSAF 255
Bruce Hoff, Michael D. Howard, David Y. Tseng
Information Sciences Laboratory, Hughes Research Laboratories
Malibu, California

Session 6a: Implementation

Representation of Missiles in ModSAF 267
Anthony J. Courtemanche, Scott E. Hamilton, Paul Monday
Loral ADS
Cambridge, Massachusetts

From CIS to Software 275
Dirk Ourston, David Blanchard, Edward Chandler, Elsie Loh
SAIC
Orlando, Florida

Implementation of a Tactical Order Generator for Computer Generated Forces 287
David R. Pratt, Howard Mohn, Robert McGhee
Department of Computer Science - Naval Postgraduate School
Monterey, California

Session 6b: Route Planning I

Unit Route Planning 295
Clark R. Karr, Sumeet Rajput
UCF/IST
Orlando, Florida

Concealed Routes in ModSAF 305
Michael J. Longtin, Dalila Megherbi
Loral ADS
Cambridge, Massachusetts

Terrain Avoidance for CGF Helicopters **315**
Stephen A. Schricker, Robert W. Franceschini, Mikel D. Petty, Tracy R. Tolley
UCF/IST
Orlando, Florida

Session 7a: Non-Military Uses of CGF

Bi-Directional Technology Transfer Between Government Applications of Computer Generated Agents and Commercial Entertainment **329**
Rich Warren, Mike Crowe, Don Shillcutt
GreyStone Technology, Inc.
San Diego, California

* **CGF Opportunities in Plowshares** **337**
Mikel Petty, Mary P. Slepow
UCF/IST
Orlando, Florida
Paul D. West
United States Military Academy
West Point, New York

Planning for Reactive Behaviors in Hide and Seek **345**
Michael B. Moore, Christopher Geib, Barry D. Reich
Department of Computer and Information Science, University of Pennsylvania
Philadelphia, Pennsylvania

Session 7b: Terrain Modeling I

Abstracting Terrain Data Through Semantic Terrain Transformations **355**
David Hille
ANSER
Arlington, Virginia
Michael R. Hieb, Gheorge Tecuci, J. Mark Pullen
Department of Computer Science, George Mason University
Fairfax, Virginia

Terrain Reasoning by Intelligent Player **367**
Gregory A. Schaper, Ashok Pandari
Department of Computer Science, East Tennessee State University
Johnson City, Tennessee

Recent Developments in ModSAF Terrain Representation **375**
Joshua E. Smith
Loral ADS
Barre, Massachusetts

Session 8a: Applications of CGF

A Method to Quantify the Application Value of Intelligent Decision Support Systems 385
Theodore Metzler, Joseph Kelly
LB&M Associates Inc.
Lawton, Oklahoma

Supporting Materiel R&D Using Linked Engineering, Constructive, and Virtual Modeling and Simulation Tools 391
John A. O'Keefe, IV
U.S. Army, Natick RD&E Center
Natick, Massachusetts
Robert McIntyre
Simulation Technologies, Inc.
Dayton, Ohio

Session 8b: Terrain Modeling II

Integrated Computer Generated Forces Terrain Database 399
Thomas Stanzione, Forrest Chamberlain
TASC
Reading, Massachusetts
Dr. Alan Evans, Cedric Buettner
SAIC
Waltham, Massachusetts

Terrain Capabilities in CCTT 411
Jon Watkins
SAIC
Orlando, Florida

Evening Plenary Session
ARPA CFOR Briefing

Implementation of Command Forces (CFOR) Simulation 423
Marnie R. Salisbury, Lashon B. Booker, David W. Seidel, Judith S. Dahmann
The MITRE Corporation
McLean, Virginia

Session 9a: Experimental Results

Experimental Conversion of the IST Computer Generated Forces Simulator from C to Ada 433
Michael A. Craft, Mikel D. Petty
UCF/IST
Orlando, Florida

Comparison of A* and Iterative Deepening A* in Graph Search 443
Clark R. Karr, Sumeet Rajput, Larry J. Breneman
UCF/IST
Orlando, Florida

Intervisibility Heuristics for Computer Generated Forces 451
Sumeet Rajput, Clark R. Karr, Mikel D. Petty, Michael A. Craft
UCF/IST
Orlando, Florida

Benchmarking and Optimization of the IST CGF Testbed 465
Stephen A. Schricker, Tracy R. Tolley, Robert W. Franceschini
UCF/IST
Orlando, Florida

Session 9b: Dismounted Infantry

Individual Combatant Development in ModSAF 479
Michael D. Howard, B. Hoff, D.Y. Tseng
Hughes Research Laboratories
Malibu, California

* **Mobility Behavior in Dismounted Forces** 487
George R. Mastroianni
U.S. Army Natick RDEC
Natick, Massachusetts
Reed W. Hoyt
USARIEM
Natick, Massachusetts
Mark J. Buller
GEO-CENTERS, Inc.
Natick, Massachusetts

A Behavioral Approach to Fidelity Requirements for Simulation of Dismounted Combatants 495
Robert T. McIntyre, III
Simulation Technology Inc.
Raleigh, North Carolina
Victor E. Middleton
Simulation Technologies, Inc.
Dayton, Ohio

✳ **Simulation of Suppression for the Dismounted Combatant** **501**
Victor E. Middleton
Simulation Technologies Inc.
Dayton, Ohio
W. M. Christenson
Institute for Defense Analyses
Alexandria, Virginia
John D'Errico
Dismounted Battlespace Battle Lab
Ft. Benning, Georgia

Session 10a: Architecture

Representing Role-Based Agents Using Coloured Petri Nets **513**
Mike Davies, Fred D. J. Bowden, John M. Dunn
Information Technology Division, DSTO
Salisbury, S. Australia

Realistic Doctrinal Behaviors in CGF Through Plurality **521**
Denis Gagné
IntelAgent R&D
Victoriaville, Quebec, Canada

A Comparison Study of Behavioral Representation Alternatives **529**
Se-hung Kwak
Loral ADS
Cambridge, Massachusetts

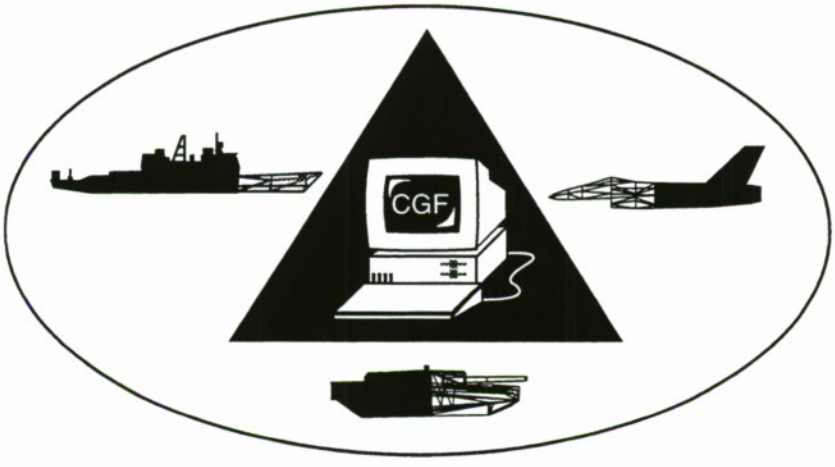
Session 10b: General Interest

✳ **The OPFOR Model in CCTT and Beyond: Applications in DIS** **543**
Penny L. Mellies
TRADOC DCSINT, Threat Support Division
Ft. Leavenworth, Kansas

✳ **Report on the State of Computer Generated Forces 1994** **549**
H. Kent Pickett
TRADOC Analysis Center
Ft. Leavenworth, Kansas
Mikel D. Petty
IST/UCF
Orlando, Florida

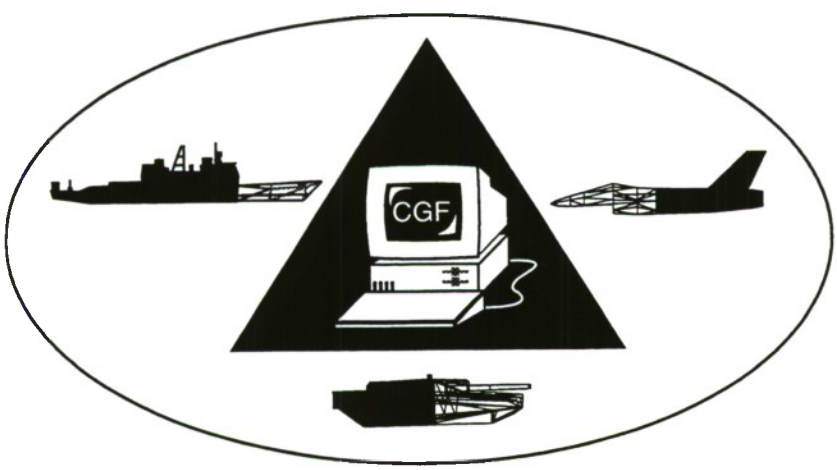
Asynchronous Rule-Based Systems in CGF **559**
Craig Williams, Paul F. Reynolds, Jr.
Department of Computer Science, University of Virginia
Charlottesville, Virginia

Author's List **567**


CGF

Session 2a: Project Status Reports

Courtemanche, Loral ADS
Davies, Information Technology Division, DSTO
Franceschini, UCF/IST
Laird, University of Michigan


CGF

ModSAF Development Status

Anthony J. Courtemanche and Andy Ceranowicz
Loral Advanced Distributed Simulation
50 Moulton St., Cambridge, MA 02138
ajc@camb-lads.loral.com
aceran@camb-lads.loral.com

1. Abstract

This paper provides an overview of capabilities recently added to the ModSAF system since version 1.0, as well as the new software development process that has been successfully used to manage the addition of 300 thousand lines of source code to the ModSAF baseline.

2. ModSAF Overview

ModSAF, or Modular Semi-Automated Forces, is the successor to the SIMNET and ODIN Semi-Automated Forces systems. It provides a modular architecture that DIS and CGF researchers can build upon and extend. The development of the ModSAF architecture was started in the spring of 1992 under ARPA/ASTO sponsorship with documentation and fielding sponsored by STRICOM. This effort resulted in the release of ModSAF 1.0 in December 1993. Subsequent development of ModSAF was jointly funded by STRICOM and ARPA/ASTO, and releases of ModSAF, starting with ModSAF 1.2 in July 1994, are continuing with a new release every 3 months. The latest version, ModSAF 1.4, was released in January 1995, and it contains over one-half million lines of software written in C.

ModSAF is intended for use as an application to support DIS studies, as a testbed to explore new CGF approaches, and as a source of components for other systems. Current ModSAF development is being driven by the requirements of the STRICOM and ARPA programs that ModSAF is supporting. Major programs driving ModSAF requirements include A2ATD (Anti-Armor Advanced Technology Demonstration) and STOW (Synthetic Theater Of War). A2ATD is intended to develop and demonstrate a verified, validated, and accredited (VV&A) DIS testbed capability to support combat and material development studies. This has resulted in the VV&A of ModSAF for A2ATD exercises. STOW has the objective of demonstrating the use of DIS for large scale exercises distributed over many sites. In this effort ModSAF is being linked to BBS, a program that models higher level command and control.

3. New Capabilities

There have been three releases since ModSAF 1.0 (ModSAF 1.2, ModSAF 1.3, and ModSAF 1.4). ModSAF 1.5 is scheduled for release in April, 1995. The following sections describe many of new capabilities that have been added during this effort.

3.1 Platoon Behaviors

ModSAF supports platoons of friendly and enemy Tank, Mechanized Infantry, Air Defense, and Combat Service Support units. The following sections list some of the new behaviors, implemented as task frames (Calder et. al. 1993) that are available to these platoons:

3.1.1 Platoon Withdraw

The Withdraw task frame moves a unit away from the enemy, and tells the unit to perform the Hasty Occupy-Position task until it receives another order. Armored vehicles withdraw in reverse if the enemy is seen; otherwise, they move in normal forward gear. If the enemy is no longer visible, an armored vehicle turns to complete the withdrawal in forward gear. Once it transitions from reverse to forward, it remains in forward even if the enemy reappears. Unarmored vehicles use forward gear to reach the withdraw point.

3.1.2 Platoon Minefield Withdraw

The Withdraw from Minefield task frame can be triggered automatically when a platoon discovers it has entered a minefield, as indicated by the detonation of mines. When executing this behavior, vehicles backtrack for a given distance and then the unit performs a normal withdraw.

3.1.3 Platoon Delay

The Delay task frame lets a platoon perform a delay maneuver. When assigning this task frame, the user enters at least one of four alternate battle positions. The Delay automatically ends when the unit reaches the last battle position. Delay divides the unit into two functional groups which perform Withdraws to the alternate battle positions in a non bounding fashion. For example, one functional group withdraws to the battle position while the second functional group stays to fire at the enemy. When the first group completes the withdraw, the second

group begins to withdraw to the same battle position occupied by the first group. Once the second group is finished occupying the battle position, the first group withdraws to the next battle position, and so on.

3.1.4 Platoon Breach

The Breach task frame divides a unit into two functional groups: an occupy group and a travel group. The occupy group performs an occupy position while the travel group moves through the area. When the travel group stops, they occupy position. The previous occupy group then moves along the same route. Mines explode if encountered, but do no damage to the vehicles during the breach. When a vehicle breaches a minefield, it can drop objects behind itself (called breach lane markers) to mark its path. This lets other vehicles see the safe route. Under the direction of the US Army Engineering School, this simplified breaching behavior is being enhanced with improved models of minefield effectiveness and countermine effectiveness, including the modeling of the Full Width Mine Plow (FWMP).

3.1.5 Platoon React To Indirect Fire and Air Attack

The React To Indirect Fire and React To Air Attack task frames are reactions that can be triggered automatically when a platoon discovers it is under attack from indirect fire or aircraft. React To Indirect Fire lets a moving ground unit respond to an artillery burst in the immediate area by accelerating. After a period of no indirect fire, the unit slows to its original pace. The React-Air task monitors enemy aircraft activity and triggers reactive events when a unit detects enemy aircraft or receives fire from them. A ground unit scatters when it sees or receives an impact packet from aircraft unless it is in a defensive battle position.

3.1.6 Platoon Attack By Fire

The Attack by Fire task frame tells a unit to advance and shoot at the enemy using alternating fire. A call for indirect fire is reported by radio at the beginning of the attack. A spot report is sent when the attack is over.

3.1.7 Platoon Overwatch Movement

The Overwatch Movement task frame divides the platoon into two functional groups; only one group moves at a time. Whenever a group is traveling along the route, the other group (referred to as the support group) is executing an Hasty Occupy Position task, monitoring the traveling section and watching for enemy vehicles. This method of travel is useful during reconnaissance missions. The Traveling Overwatch task frame also divides the unit into two functional groups; a traveling group that moves out in front and a support group that follows behind.

3.1.8 Platoon Assemble

The Assemble task frame instructs a moving platoon to form a coil formation and then stop moving.

3.1.9 Platoon Hasty Occupy Battle Position

The Hasty Occupy Battle Position task frame is perhaps the most complex ground task frame. When the user assigns this frame to a platoon, he specifies a battle position and an engagement area, which may be a point, line or area. Based on the battle position and the number of subordinates, ModSAF calculates both the number of vehicles per segment (the battle position consists of one or more line segments) and the battle areas (areas where each vehicle searches for cover positions). ModSAF assigns the unit subordinates positions from one end of the battle position to the other in an order that prevents vehicle crossover when the vehicles travel to their positions. When the unit arrives at the battle area, ModSAF places the vehicles in covered (hull defilade) firing positions along the battle position line. ModSAF considers a vehicle's limits for angles of elevation and depression when selecting a good cover position. For example, a vehicle is not placed at a location where underlying terrain prevents the gun from being physically pointed at the enemy. If that location is behind a tree line or building and no cover location is found, the vehicle moves to a concealed (partially hidden) location behind the tree line or building. The vehicles also move to areas where they can maintain visibility to the engagement area. Target Reference Points (TRPs) can be supplied by the user or automatically computed. Enemy vehicles detected within the sector of fire determined by the left and right TRPs are considered first priority targets.

Units executing this task frame may optionally execute movement to an alternating fire position. When a vehicle detects that it is receiving accurate enemy fire, it may move from its primary firing position to an alternate hull-defilade firing position. When it does this, it first backs up to a fully hidden position to mask the new location that it is about to take up. In addition, vehicles have enemy awareness during the execution of this task frame. As enemies move in and out of the engagement area, vehicles will recalculate the location of their covered primary and alternate firing positions.

3.2 Company Behaviors

ModSAF supports companies of friendly and enemy Tank, and Mechanized Infantry units. The following sections list some of the behaviors of companies.

3.2.1 Company March and Roadmarch

March in formation and Road March task frames can be assigned to companies. These behaviors decompose the company order into platoon march orders for the subordinate platoons. During movement, the extra vehicles (such as the company commander, XO, and others) are functionally organized into the platoons that stay in formation. At the end of the route, the extra vehicles (the leaders) revert to the task organization that the unit started with.

3.2.2 Company Attack

In the Company Attack task frame, the company advances in line formation to the objective with fire permission set to "free". When the company reaches the attack objective its vehicles occupy a battle position at that location facing the direction of the attack. The company can also stop before reaching the objective and occupy a position if the number of casualties is too high. This task frame is similar to the Assault task frame for ground platoons.

3.2.3 Company Withdraw

When assigning the Company Withdraw task frame to a ground company, the operator specifies to which point the company can withdraw. ModSAF uses this point to generate final withdraw points for each platoon. Vehicles in the company that are not part of a platoon are functionally organized to a platoon. The platoons withdraw one at a time. When the withdrawing platoon occupies position, the next platoon withdraws, and so on, until the platoons reach their final points.

3.2.4 Company Assembly Area

The Company Assembly Area task frame tells a company to advance to, and then occupy, an assembly area. When the operator assigns this task frame, he specifies a position that a unit can assemble around. The assembly area is a circle whose center is the point supplied. Its default radius is 700 meters. The company uses platoon battle positions which provide 360 degree coverage. ModSAF creates TRPs along with the battle positions.

3.2.5 Company Hasty Occupy Battle Position

To use the Company Hasty Occupy Position task frame, the operator supplies a battle position and three TRPs: a left TRP, right TRP, and engagement area TRP. ModSAF divides the battle position among the platoons in the company. ModSAF then creates left and right TRPs for each platoon. The company commander and executive officer are functionally organized into one platoon each. Each platoon moves to its battle position and receives its calculated left and right TRP with the unmodified engagement area TRP.

3.3 Situation Awareness and Reports

All ModSAF units maintain situation awareness. Individual vehicles apply the NVL acquisition model (Courtemanche and Monday 1994) to detect enemy vehicles. The detections of individual vehicles in a unit are fused together by platoon and company behaviors. Platoon leaders or company commanders use this information to construct a situation awareness overlay which contains the locations of observed enemy fused into platoon- or company-sized clusters. The operator has access to this graphical overlay maintained by each unit under his control and can examine the situational awareness of that unit.

The clusters of fused enemy observations drive the sending of reports to the operator. The types of reports supported include contact reports, spot reports, and shell reports. These reports are available in a message log at the operator's workstation. In addition, these reports can also be sent as digital Inter-Vehicular Information System (IVIS) messages which can be received by the occupants of manned simulators.

3.4 Individual Combatants

ModSAF now supports teams of individual combatants. Two-man Stinger teams, which are represented as two separate DIS entities, can engage aircraft with the Stinger missile, while two-man Javelin and Dragon teams can engage tanks with anti-tank missiles.

3.5 Fire Support

ModSAF supports batteries of enemy and friendly mortar, howitzer, and MLRS units which are capable of delivering indirect fire. Command and control between Fire Direction Centers (FDCs) and the fire support units are modeled via the exchange of ASCII encoded messages transmitted via DIS Signal PDUs. FASCAM delivered minefields are also supported. Counter Battery Radar can detect enemy artillery and generate fire orders to engage the inferred location of the enemy artillery units.

3.6 Air Defense

In addition to the Stinger teams, platoon-sized units provide air defense in ModSAF. Threat vehicles such as ZSU 23/4, SA-9, and 2S6 are supported, as well a friendly Avenger unit which is based on HMMWV trucks equipped with Stinger missiles. A Ground Based Sensor (GBS) can provide early warning of threat aircraft to the Avenger unit.

3.7 Combat Service Support

Platoons of combat service support vehicles give ModSAF the ability to execute towing, resupply, and repair behaviors. Towing is accomplished via the experimental entity handover protocol proposed for DIS 2.1. Resupply and repair is accomplished via the DIS 2.0.3 resupply and repair PDUs. Both service-station and tailgate resupply is supported, as well as the cross leveling of supplies between vehicles.

3.8 FWA

ModSAF supports flights of A10, F16, and SU-25. ground attack aircraft. These aircraft can fly in formation using a contour flight mode and can execute attacks on ground targets with missiles, guns, and bombs. The ModSAF operator can specify various types of attack geometries, deliveries and entries.

3.9 RWA

ModSAF supports flights of enemy and friendly rotary wing aircraft, including AH-64D Apache and RAH-66 Commanche. Remote laser designation of targets for the Hellfire missile is supported. Laser designation makes use of the DIS Laser PDU. RWA units can follow a route or orbit using low level, contour, or nap of earth (NOE) movement techniques. RWA units can Assemble, as well as execute a Hasty Occupy Battle Position task frame. RWA units can perform attacks using hover fire and running fire techniques.

3.10 Specialized Systems

Many specialized weapon systems are supported in ModSAF. The following sections describe some of these systems and their behaviors.

3.10.1 LOSAT

To ensure compatibility for the A2ATD exercises, the Line of Sight Anti-Tank (LOSAT) fire unit modeled in ModSAF is representative of the "AGS quick fix" LOSAT DIS Crew Station Simulator (DISCSS). This LOSAT fire unit consists of a hybrid chassis with Armored Gun System (AGS) dimensions with M2 mobility and vulnerability. AGS-LOSAT sensor configurations, AGS-LOSAT launcher/weapons configurations and AGS-LOSAT fire controls are supported. The sensors represented include the primary sensor (FLIR & TV), IR secondary observation sensor, and the driver and commander's vision blocks. The weapon's cage assembly consists of two turret mounted missile pods of six missiles each (modeled as one cage of 12) and a coaxial mounted 7.62mm machine-gun for local security purposes. AGS-LOSAT fire controls are capable of autotracking and engaging in groups of up to three targets. The missile is representative of the Kinetic Energy (KE) missile which is utilized in the AGS-LOSAT variant.

The basic units provided are the section and platoon. Each section consists of two LOSATs. In addition to the basic units, a LOSAT augmented M2 company is provided for experimentation. This unit consists of a normal M2 reinforced company which has been augmented by an appropriate slice of "Echo Company" ("Echo Company" is the anti-armor company in a mechanized infantry battalion). In this case, two sections of LOSAT have been aggregated into the M2 company so that they will maneuver as a unit together. Movement and tactics are accomplished by the taskframes which were previously available to ground units in ModSAF.

3.10.2 NLOS

ModSAF supports an NLOS vehicle which consists of a HMMWV with a NLOS missile launcher. The NLOS vehicle can receive contact reports from ground forces or a UAV to engage non-line-of-sight targets.

3.10.3 UAV

ModSAF supports a simplified unmanned air vehicle (UAV) which can observe the battlefield and provide targets to systems such as NLOS.

3.10.4 STAFF

The Smart Target Activated Fire and Forget (STAFF) munition is a 120mm main gun round being developed for the M1A1 and M1A2 man battle tanks. STAFF is fired from the main gun at the intended target. At some distance before the round reaches the target, a radar in the front of the round begins scanning the forward area. When a target enters the radar field of view, the round tracks the target and fires a submunition which is the primary kill mechanism of the round. The seeker and submunition provide the capability for STAFF to hit the target when the body of the main round misses the target. ModSAF implements this munition via an Army Materiel Systems Analysis Activity (AMSAA) approved methodology.

3.11 Manned Simulator Interoperability

In order to participate in DIS exercises, it is critical that ModSAF interoperate with manned simulators. The following are examples of new behaviors developed to support manned simulator interoperability.

3.11.1 Follow Simulator

The Follow Simulator task frame allows a platoon to follow a simulator. The simulator and platoon are in

formation together. If the simulator deactivates or is incapable of movement, the platoon will occupy a position. If the simulator reactivates, and if it is close enough to the platoon, the platoon will rejoin it, otherwise the platoon will continue to occupy a position. If the platoon is tasked away and then the Follow Simulator task frame is resumed, if the simulator is close enough to the platoon the platoon will rejoin it, otherwise the platoon will occupy a position. Cue fire allows vehicles that are performing a follow vehicle to fire when that vehicle fires.

3.11.2 Digital Communications

A number of efforts are underway to enhance ModSAF for use in experiments exploring digitization of the battlefield. As stated in section, 3.3, ModSAF units can supply digital reports to manned simulators. In addition, messages are being defined to allow digital communications between manned AH-64D Apache Longbow simulators, RAH-66 Commanche simulators, and ModSAF RWA units. These messages include target coordination.

3.12 Mine/Countermine

The Improved Mine/Countermine Delivery Order is extending ModSAF with improved models of minefield effectiveness and counterobstacle effectiveness, including minefield breaching and bridge laying. This work has included the development of the Full Width Mine Plow (FWMP) and Armored Vehicle Launched Bridge (AVLB). The ability to generate dynamic ditches in the terrain is being explored, and the AVLB can be used to allow armored units to overcome these obstacles.

3.13 Phenomenology

The ARPA/TEC sponsored Project Phenomenology and the Dynamic Virtual Worlds projects have extended ModSAF with environmental effects such as battlefield smoke, temperature, illumination, and rain (Schaffer 1994). This work has included the addition of tactical behaviors such as launching smoke grenades during a platoon withdraw task frame. Additional work is proceeding on improving these environmental models and adding signal flares and vehicular dust.

3.14 User Interface & Missions

ModSAF has maintained a unified user interface for operator control (Ceranowicz et. al. 1994). Operators are able to construct and monitor missions via an execution matrix paradigm. Immediate commands can be rapidly issued via an Immediate Intervention interface.

4. Software Development Process

The capabilities that have been added since ModSAF version 1.0 have required an unprecedented amount of development parallelism while maintaining system integrity and software quality. The following sections describe the software development process that has succeeded in managing this effort.

4.1 Distributed Development Team

Figure 1 shows the locations of the development teams that have contributed to the ModSAF baseline since ModSAF version 1.0, the sites involved with VV&A (AMSAA and TRAC), as well as the sponsoring organizations (ARPA and STRICOM). The use of a number of subcontractors has been motivated by a desire to leverage existing ModSAF and/or subject matter expertise, as well as requirements to accelerate development schedule to meet experiment requirements.

In addition to development that is directly integrated into the ModSAF baseline, many ModSAF users are extending ModSAF under independent efforts.

4.2 Frozen Interfaces & ECOs

The management of such a highly distributed software development team requires a controlled software engineering process. With so many different teams that are developing and modifying software, there is a tremendous risk of software chaos. If one team were to modify software that another team depends on, there is no guarantee that the combined efforts of each team will be compatible.

It was decided early on that the only approach to solve this problem would be to "freeze" all existing public ModSAF software interfaces. The ModSAF coding standards make clear distinction between which interfaces are public and which are private. The ModSAF software is divided into a large number of layered software libraries. Each library advertises its public interfaces via header files and on-line documentation. These interfaces include data structures, global variables, argument prototypes for public functions, and return values of public functions. These interfaces can be used by other libraries that are layered above this library. It is these public interfaces which are said to be frozen in ModSAF. They are unable to be changed without an orderly process, and each software development team can depend on these interfaces as they write new software capabilities.

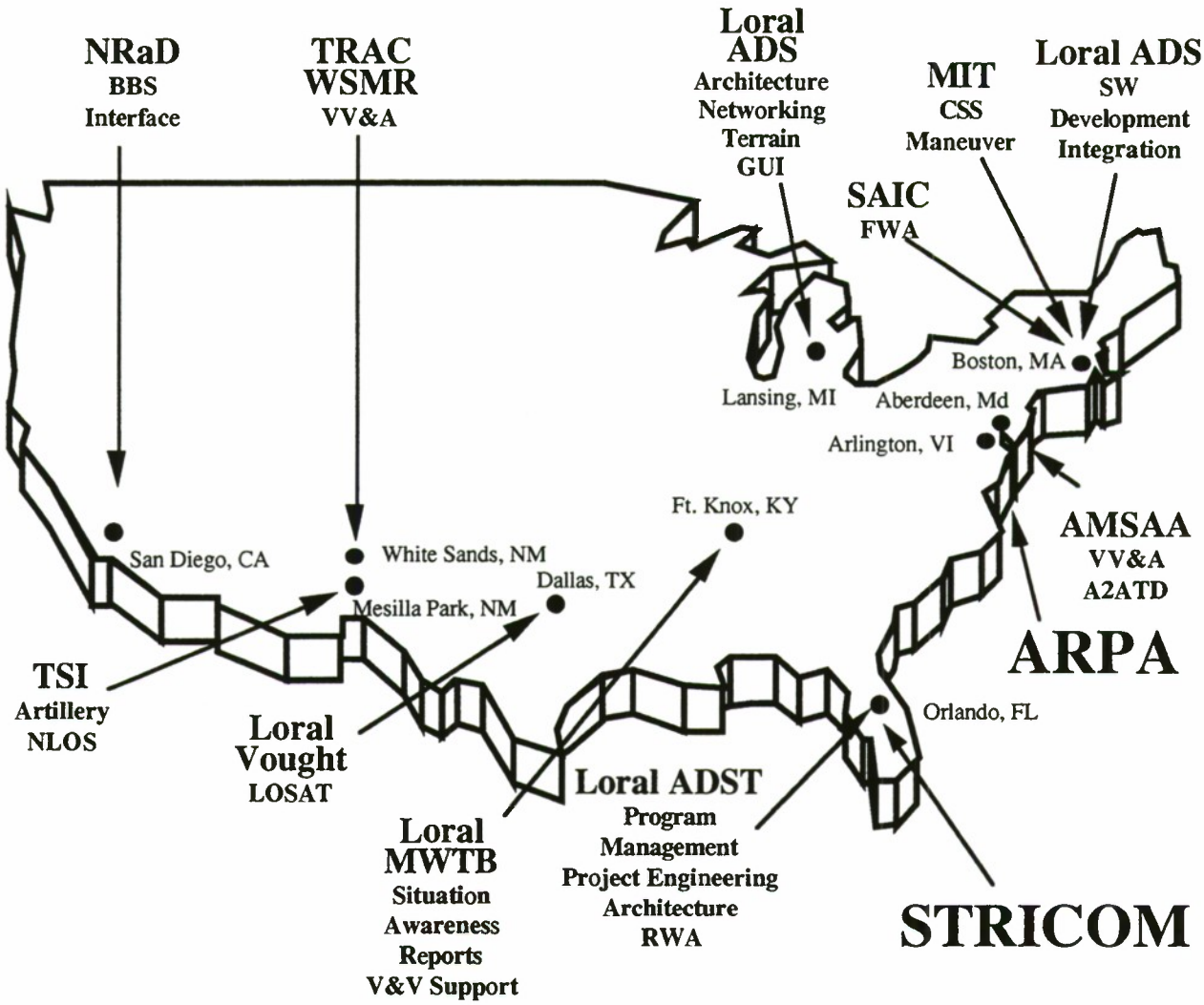


Figure 1: ModSAF Development Sites

Of course, the existing interfaces in ModSAF were far from being perfect when the freeze went into effect. For this reason, an Engineering Change Order (ECO) process was established to control the required changes to these interfaces. Each software developer that wishes to change a public software interface must publish the desired changes in advance via a formalized ECO request form. This form documents the nature of the change, a justification for making the change, what existing libraries might be effected by the change, and when the change is proposed to be integrated into the ModSAF baseline. The ModSAF Project Engineer is responsible for collecting, reviewing, and coordinating all ECOs between the software development teams. ECOs must be approved and distributed to all development teams prior to any modification of a public interface being integrated into the ModSAF baseline.

It is typical for design discussions between interested parties to precede the request and approval of ECOs. Software designs are published by developers and distributed to all software development teams via electronic mail. This allows all developers to comment on designs prior to implementation. ECO requests are mailed electronically to the Project Engineer, and are mailed electronically to all developers once approved.

A simple database of all ECOs is maintained by the Project Engineer, and the status of each ECO (pending approval, approved, integrated) is carefully tracked. A listing of all ECOs since the last release is published in the *Version Description Document* (VDD) that accompanies each software release.

4.3 Formal Integrations

In order to manage the rapid development between multiple software development teams working in parallel, an integration facility has been set up to act as the developmental configuration management site. The integration facility is co-located with the majority of ModSAF software development personnel in the Loral ADS headquarters in Cambridge, MA. Over the course of the last year, the specific hardware makeup of the integration facility has varied, but it currently includes a GT100 based Stealth, 3 SGI workstations, 1 Sun Sparc workstation, and 1 Mips workstation connected via a private Ethernet. One SGI workstation acts as the primary configuration management host, and runs the Concurrent Versions System (CVS) configuration management tool.

CVS provides the support for the rolling baseline approach that characterizes current ModSAF development. Multiple independent development teams work in parallel from different baselines to minimize the development schedule. Figure 2 shows the types of overlapping development and integration that are possible.

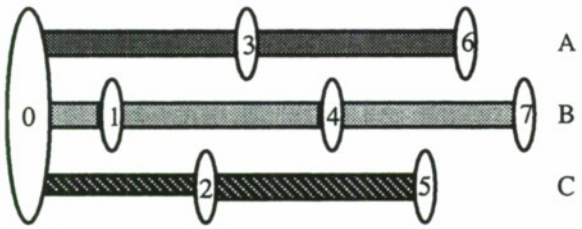


Figure 2: Rolling Baseline, Developer Perspective

This diagram shows three development teams, lettered as A, B, and C, with their activity represented as shaded bars. Each team originally starts off development using a common baseline release, in this case labeled 0. Team B is the first to integrate new capabilities into the baseline, at the integration labeled 1. Once that integration is complete, Team B continues to do development, however the new development is based on the newly integrated capabilities. Team C integrates next, and they merge new capabilities with the new baseline created at the previous integration. Once Team C completes their integration, they continues to do new development, this time based on the newly merged baseline. Team A finally integrates at the integration labeled 3, and they must merge new capabilities with the baseline established by Team C. At any given point in time, there are 3 active baselines under development. In one baseline development may be just starting, while in another, development may be ready to be integrated into the main line.

To support the integration facility, a dedicated integration team manages all the integrations. The integration team is responsible for maintaining the main baseline and merging in new capabilities. From the perspective of the dedicated integration team, the baseline looks like Figure 3.

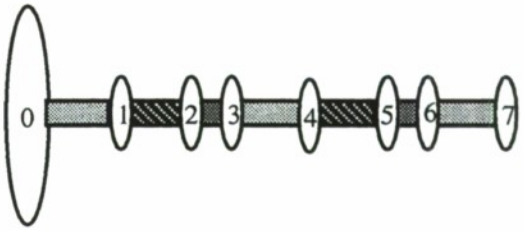


Figure 3: Rolling Baseline, Integration Team Perspective

The integration team sees the main baseline being constantly updated with new capabilities. The integration team uses the CVS tool to help automate the 3-way merge between the current baseline, the original baseline that the integrating team's software was based on, and the new updated baseline that the integrating team brings to the integration facility.

Without proper testing, the rapid and constant introduction of new capabilities into the main baseline could jeopardize the quality of the ModSAF software. In order to maintain quality, two types of testing are done at each integration. First, capabilities testing is performed to guarantee that only robust completed capabilities are actually integrated into the system. The capabilities tests demonstrate that what is about to be integrated actually works. Capabilities tests will typically consist of interactive instructions for the integration team to follow to exercise a new capability. The capabilities tests are typically designed by the head of the software team developing the capability. and the tests are released as part of the ModSAF software distribution.

The second type of testing that occurs at each integration is regression testing. Regression testing attempts to guarantee that no previously existing functionality has been damaged by the integration of the new capabilities. Regression testing consists of well defined scenarios with repeatable results, as well as portions of previously accepted capabilities tests. Because of the great breadth of capabilities present in ModSAF, it is impossible to perform a complete regression test in the short time allocated for an integration. Automated tests, as described in a companion paper (Monday and Perneski 1995), maximize the amount of testing done on the system.

In addition to the tests performed at each integration, system benchmarking (Vrablik & Richardson, 1994) is done to measure any changes in system performance introduced by the new capabilities. Changes in performance are closely monitored, and any sudden decrease may force a reevaluation of the implementations chosen for the new capabilities.

The serial nature of the integration facility makes it the critical resource for software development. The integration facility is generally always in use. Each integration typically lasts 2-3 days, and there are usually 2 integrations per week.

4.4 Software Releases

ModSAF releases currently follow a 3 month cycle. Fortunately, due to the constant amount of testing going on in the integration facility, a software release is not as traumatic an experience as it once was. The software is completely compiled from scratch at each integration, so there is rarely any concern about the ability to "cold start" a software build.

Two weeks prior to a release, the ModSAF software goes through an extensive period of testing. Teams are assigned to test each functional area, such as ground vehicle tasks, artillery, rotary wing, etc. During the course of testing, software defects are logged into the defect tracking system, and testing reports are generated.

During the testing process, periodic reviews of the list of defects are performed to identify those defects which should and can be addressed prior to the release. Engineers fix the highest priority problems and integrate them in mini "bug-fix" integrations during the testing period. The last "bug-fix" integration slot is typically a few days prior to the final release. Any problems identified after this integration are noted in the VDD.

In addition to the preparation of the software, documentation is prepared for each release. All the on-line documentation that comprises the *Programmer's Reference Manual* (PRM) is converted into PostScript format for placement on the release tape. Also, final versions of the VDD are assembled, including all the open and closed defects as well as all the ECOs. Two documents are under constant update through the development process. The first is the *Users Manual*, which describes how to run the ModSAF system. The second is the *Functional Description* which documents all the ModSAF capabilities. Final versions of these documents are prepared for the final release.

During the release process, Quality Assurance (Q/A) representatives review the process for adherence to the established processes and test plans. The final release software, on-line documentation, and hardcopy documentation is delivered to Q/A for verification and duplication prior to distribution.

4.5 Defect Tracking

A defect tracking system called GNATS is used during the entire ModSAF development process. GNATS is distributed by the Free Software Foundation. It partially automates the tracking of problems by organizing problem reports into a database, notifying responsible parties of suspected bugs, allowing support personnel to edit, query, and report on accumulated bugs, and providing a reliable archive of problems and fixes in the system. The main component of GNATS are problem reports (PRs) that are generated by ModSAF users as well as ModSAF developers. PRs are organized in to categories, prioritized, and distributed to responsible engineers.

The list of open and closed PRs is published as part of the VDD for every ModSAF release.

4.6 Software Distribution and User Support

Once released, ModSAF software is distributed by the Tactical Warfare Simulation and Technology Information Analysis Center (TWSTIAC), which is affiliated with the University of Central Florida's Institute for Simulation and Training (IST). The TWSTIAC has a general charter of providing scientific and technical information and support services to government, industrial, and academic communities in the area of modeling and simulation. As part of this function, the TWSTIAC distributes software of general interest to the DIS community. Individuals and organizations with official government sponsors can contact the TWSTIAC to request distributions of ModSAF software.

In addition to supporting distribution via the TWSTIAC, IST maintains a ModSAF electronic-mail reflector which is used as a forum for ModSAF users to ask questions, share insights, and have access to the main body of ModSAF developers. Access to this mail reflector is available to anyone in the DIS community. Under the direction of STRICOM, Loral engineers and support personnel coordinate and answers questions posed on this reflector.

4.7 Software Statistics

Figure 4 shows the ModSAF system software growth in lines of code and software libraries since ModSAF version 1.0.

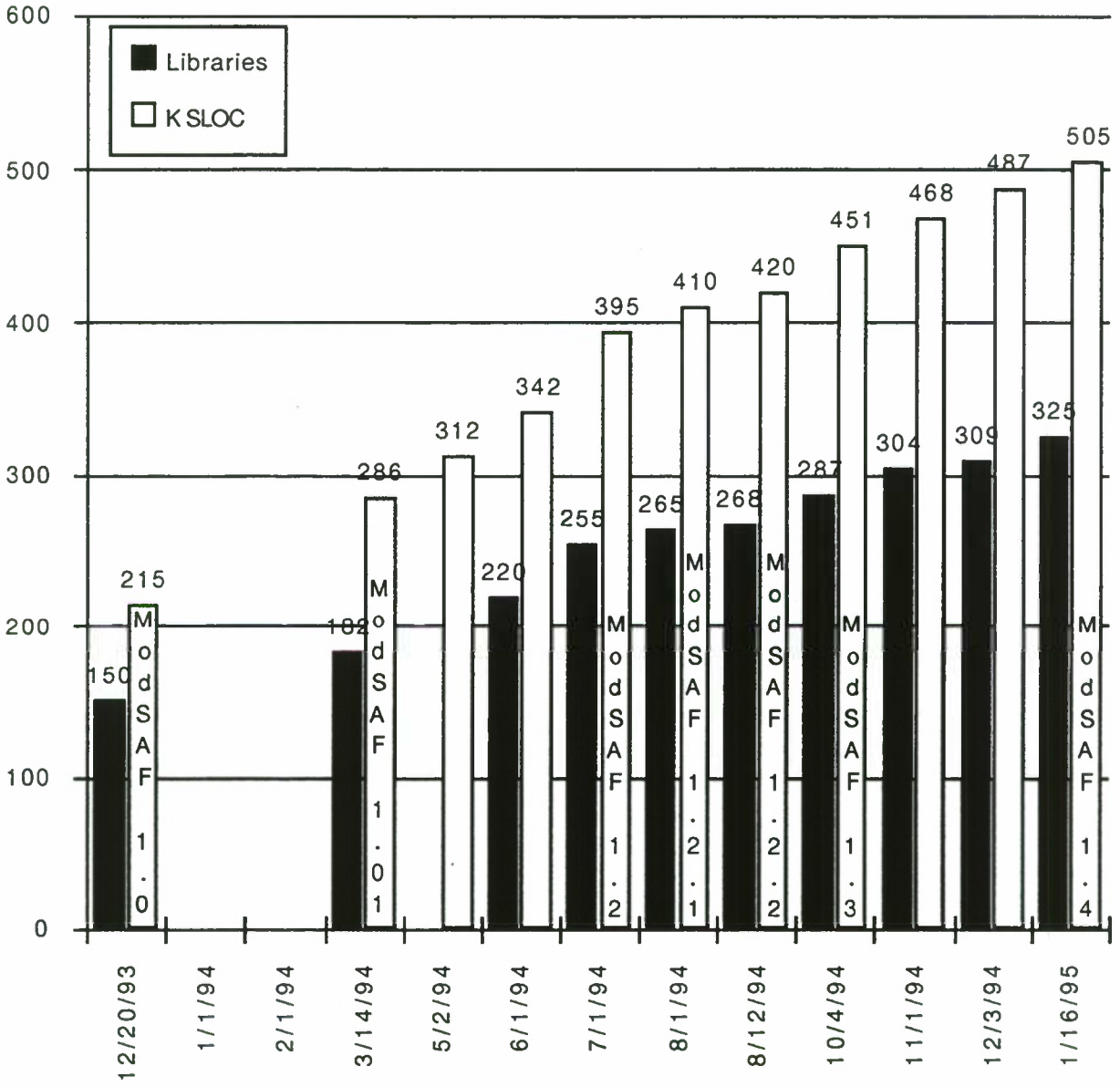


Figure 4: ModSAF Software Statistics

5. VV&A & A2ATD

As part of the A2ATD program and other Delivery Orders, ModSAF is continuing to undergo verification, validation, and accreditation (VV&A). A description of the types of V&V activity that has occurred in ModSAF in support of the A2ATD program can be found in Thomas (1995).

Under the Improved Mine/Countermine Delivery Order, AMSAA, TRAC, and the US Army Engineer School, Ft. Leonard Wood, will be performing verification and validation of improved breaching behaviors, mine effectiveness, and countermine equipment being developed for ModSAF.

6. Conclusions and Future Work

There is no immediate end in sight to the amount of additional future development in ModSAF. Work is continuing in a BBS-ModSAF linkage. Additional work in Combat Support/Combat Service Support (CS/CSS) is about to begin development. Additional work in Mine/Countermine development is proceeding. Linkages with other constructive simulations such as Eagle (Calder & Evans, 1994), and CBS are under development. Future efforts to model and incorporate additional Communication Command & Control (C3) are likely.

Additional work under the sponsorship of ARPA is using ModSAF as a tool for the development of next generation networks, including multicasting, as well as next generation protocols. This is in support of ARPA's STOW efforts (Smith et. al. 1995; Calvin et. al 1995).

As the ModSAF software has been distributed to a large portion of the DIS community, a number of researchers and experimenters have been using and extending ModSAF software. The management and maintenance of the many resulting efforts will prove to be a critical challenge in the future of ModSAF.

7. Acknowledgments

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058.

8. References

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., and Ceranowicz, A. Z. (1993). "ModSAF Behavior Simulation and Control", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 347-356.

Calder, R. B., and Evans, A. E. (1994). "Construction of a Corps Level CGF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 487-496.

Calvin, J., Seeger, J., Troxel, G., & Van Hook, D. (1995). "STOW Real-time Information Transfer and Networking Architecture", *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL: Institute for Simulation & Training.

Ceranowicz, A., Coffin, D., Smith, J., Gonzalez, R., and Ladd, C. (1994). "Operator Control of Behavior in ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 9-16.

Courtemanche, A. J., and Monday, P. (1994). "The Incorporation of Validated Combat Models into ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 129-140.

Monday, P., and Perneski, J. (1995). "The Use of Automated Regression and VVA Testing in ModSAF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training.

Schaffer, R. (1994). "Environmental Extensions to ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 17-23.

Smith, J., Russo, K., Schuette, L (1995). "Prototype Multicast IP Implementation in ModSAF", *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL: Institute for Simulation & Training.

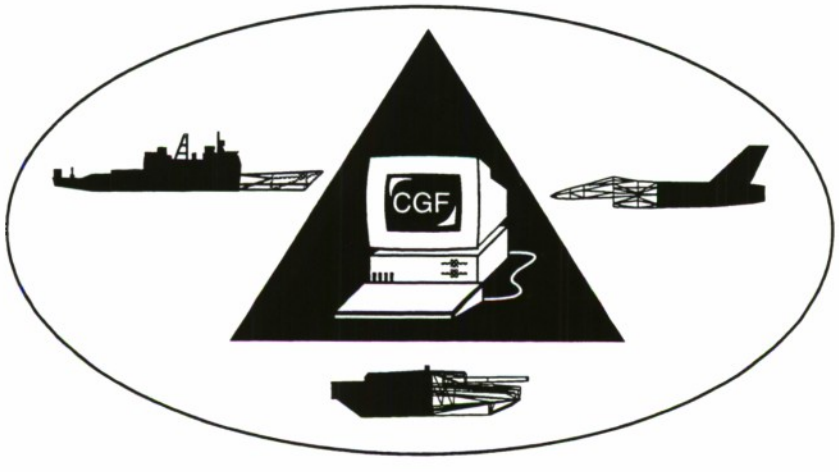
Thomas, J. Jr. (1995). "Verification and Validation of Modular Semi-Automated Forces (ModSAF) in Support of A2ATD Experiment 1", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training.

Vrablik, R., and Richardson, W. (1994). "Benchmarking and Optimization of ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 25-33.

9. Biographies

Since graduating from MIT with a S.M. in Electrical Engineering and Computer Science in 1987 and a S.B. in Electrical Engineering in 1986, Mr. Courtemanche has worked in the Multiprocessor Lisp group at Bolt Beranek and Newman (BBN) and in the SAF group at Loral Advanced Distributed Simulation (formerly BBN Advanced Simulation). At Loral, he is a Senior Software Engineering Specialist and has been Project Engineer for the ModSAF System Development Delivery Order. Mr. Courtemanche has been a major contributor to the architecture, protocols, weapon systems modeling, and targeting behaviors in ModSAF, Odin, TCE, and SIMNET SAF. He works out of the Loral ADST office in Orlando, Florida.

Andy Ceranowicz in the manager of the Semi-Automated Forces group at Loral Advanced Distributed Simulation. He has been working on Distributed Interactive Simulation and Semi-Automated Forces since 1986 when he joined BBN to work on the ARPA Simulation Networking (SIMNET) Project. Prior to his work at Loral, he was a member of the technical staff at Draper Laboratory working on expert systems and GPS applications. Dr. Ceranowicz earned his PhD in control theory from the Ohio State University.


CGF

The Distributed Interactive C3I Effectiveness (DICE) Simulation Project: An Overview

Mike Davies and Carsten Gabrisch
Information Technology Division
Defence Science and Technology Organisation, Salisbury
PO Box 1500
Salisbury, South Australia, 5108.

1. Abstract

The Defence Science and Technology Organisation (DSTO) is currently tasked by the Headquarters Australian Defence Force (HQADF) to develop tools, through modelling and simulation, for effectiveness studies of Command, Control, Communication and Intelligence (C3I) systems. Such tools must allow for the study of C3I systems at the strategic, operational and tactical levels including all services and joint forces. The primary tool being developed is the Distributed Interactive C3I Effectiveness (DICE) simulation in which human players are complemented by artificial agents. The DICE simulation environment will provide a means of interfacing to lower level battlefield simulations and war games which are used to represent the overall military mission, operation or battle. The impact of C3I aspects on the overall mission will be used to gauge C3I system effectiveness. This paper gives an overview of the DICE simulation project and associated activities.

2. Introduction

As stressed in the Australian Defence White Paper of 1994 (AGPS 1994), effective Command, Control, Communication and Intelligence (C3I) of Australia's forces is fundamental to the successful conduct of the Australian Defence Force (ADF), in any conflict or peacetime activity. Accordingly, there is a drive to identify and remedy the weaknesses of existing C3I systems and to specify and work towards goal architectures for the future. Having the ability to study the effectiveness of existing and future military C3I systems is essential to this process. The approach adopted to carry out such studies is dependent on a number of factors, including the level of conflict to be considered, the availability of technology and the cost of the implementation (Fogg 1993). Information Technology Division (ITD) of the Defence Science and Technology Organisation (DSTO) is sponsored to develop modelling and simulation tools to enable such studies to be conducted.

A typical C3I system structure for the defence of Australia might involve elements of the three services and accommodate conflict at all levels. The strategic level embraces the higher echelons of the military and political organisations concerned and hence addressing this level requires addressing decision-making at lower (operational and tactical) levels also. The C3I architecture can be pictured as a complex network of nodes and links. The nodes are typically centres of decision making; information processing or filtering; information transfer; or combinations of these. The links are the inter-node communication channels that transmit information of many forms. In a time of conflict, the C3I system might be stimulated by intelligence concerning the detection of potentially hostile enemy activity. This would consequently cause the generation and passage of internal information that might result in changes in readiness and maybe, the deployment of reaction forces. To study the effectiveness of military C3I systems requires analysis of the impact of C3I procedures and technologies on the overall military mission concerned. The term *mission* is used here to describe, for example, an operation, battle or exercise.

The requirement on ITD is not to develop tools specific to any particular military service or level of conflict, but rather to create a general purpose suite of tools, specific instantiations of which could be used to address any particular study at hand. It was decided that the main means of achieving this suite of tools and the associated expertise would be through the process of developing an interactive simulation with some capability for remote participation. This paper gives an overview of the aims and current status of the Distributed Interactive C3I Effectiveness (DICE) simulation and associated developments.

3. General Requirements

The nucleus of the software tools being developed under this task is an interactive simulation with some distributive capability that enables participation by a number of possibly remotely located human players. Players might represent individual, or groups of, commanders in a C3I system or an aggregated representation of some other C3I system entity. The interactive nature of the DICE simulation will allow the decision-making practices of real commanders to be injected and accommodated. Such human players will need, however, to be complemented by a number of artificial ones (artificial agents) in a manner whereby the two types can communicate.

Battle simulations and possibly war games will be employed to address the tactical levels and will need to be interfaced with the main simulation in order for the represented C3I system to have impact on the military mission concerned and hence allow evaluation of the system's effectiveness. Matters that need to be addressed by such interfacing include the blending of spatial and temporal based battlefield models with the message or information flow based C3I simulation.

For flexibility, the simulation needs to be capable of running in interactive and non-interactive modes and have variable execution rate allowing real-time and non real-time execution. A scenario generation capability is required such that specification of a scenario can be easily achieved by analyst-assisted military personnel. An extensive analysis capability is required to allow effectiveness evaluation to be conducted; this might include a replay facility and history and review capability.

4. Simulation Structure and Development

A typical scenario to be represented by the DICE simulation can be regarded as centred about a complex set of nodes and links that represents the central C3I network. The real and artificial players in the DICE simulation generally form the nodes of this central network. In real C3I systems, communications between nodes can take many forms including both formatted and unformatted messages; tables of data; graphical displays; and video images. In the simulation, all forms of communication are represented by the passage of formatted textual messages which either bear a direct resemblance to a military message, or accompany or summarise information of a different form. Having a standard language, understandable to both humans and machines, for information exchange is also a major requirement in command and control systems of the ADF. The Australian Defence FORmatted Message System (ADFORMS) is the agreed standard for the ADF and this standard has been chosen as a foundation in the DICE project.

The central network can be considered to be surrounded by an external environment or node that encompasses any aspects that are not explicitly represented in the central network but which, nevertheless, form important contributions to the scenario being addressed. What lies outside the central network and what lies within depends on the depth and breadth of the scenario being simulated. The conceptual external node embraces such aspects as enemy activity; battlefield information; and sensor information which might be represented by individual models, simulations or simple look-up tables. Communication between the central environment and the external node is again achieved through the use of messages with the external node injecting stimuli into the central network of the scenario.

The overall structure, requirements and developments to date of the DICE simulation are best presented with reference to Figure 1 which is a breakdown of the functional areas of the DICE simulation environment. Indicated by the uppermost row of this figure are the *players* in the simulation, namely the simulation controller or analyst; artificial agents; and human commanders. *Peripheral units* in the DICE environment include any war games and battle simulations that may be employed plus the command support systems (CSS) that may be required by the human players. Players plus peripheral units are the nodes in the overall DICE simulation, ie the central network plus the external environment. The simulation is primarily being developed in the ANSI 'C' programming language using Sun SPARCstations. The Ingres relational database management system and associated utilities also feature strongly, along with the declarative language Prolog. The functional diagram is addressed by the following sections.

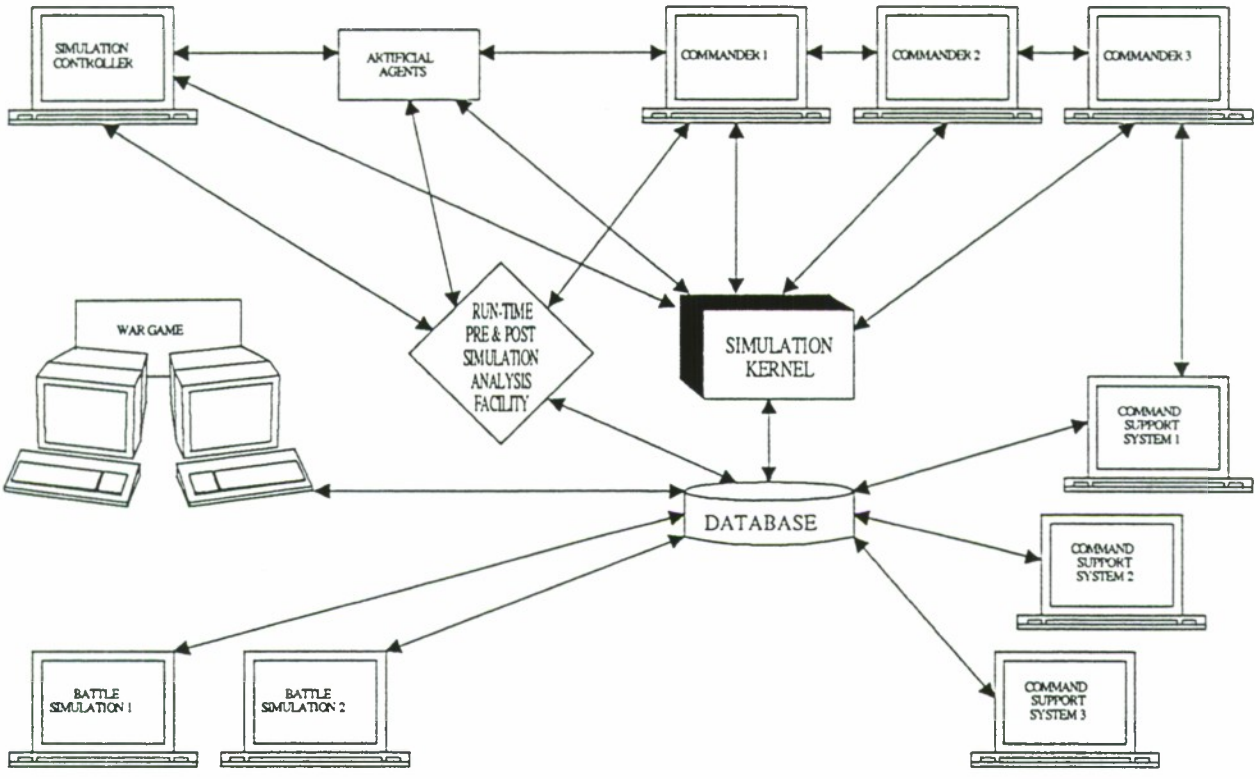


Figure 1: Functional diagram for DICE simulation environment

4.1 Simulation kernel

The simulation kernel is the main event-stepping engine of the DICE simulation. The current simulation is designed around two main events: *message submission*, involving submission of a message by a node for transmission; and *message reception*, concerning the receipt of a message by a node (Davies 1993). The kernel controls time synchronisation and execution rate of the distributed processes and can be configured such that the simulation can run in real or non-real time. The simulation is capable of being paused, advanced and resumed as required by the simulation controller.

The kernel is centred around an Ingres database and is capable of generic communication with all nodes in the simulation. The communication architecture associated with the simulation is illustrated in Figure 2. Each node has an associated mailbox through which it receives incoming messages; outgoing messages are placed directly on the simulation event queue for mailing to the intended recipient. The mailboxes are Ingres database tables which, through the use of database event features that Ingres provides, allow event-driven message reception and time-synchronisation. Interfacing between the main DICE simulation and the peripheral units is achieved through the use of Peripheral Unit Interfaces (PUI), one for each peripheral unit.

It should be noted that the communication architecture outlined in this section is intended for the DICE simulation alone and limited to locally distributed processes. Remote participation by human players and the interfacing to peripheral units that are distributed geographically are expected to be achieved through observation of international Distributed Interactive Simulation (DIS) protocols.

The simulation is typically initiated by pre-scheduled external stimuli and ends at a pre-set instant; when there are no further events in the event queue; or following a decision by the simulation controller.

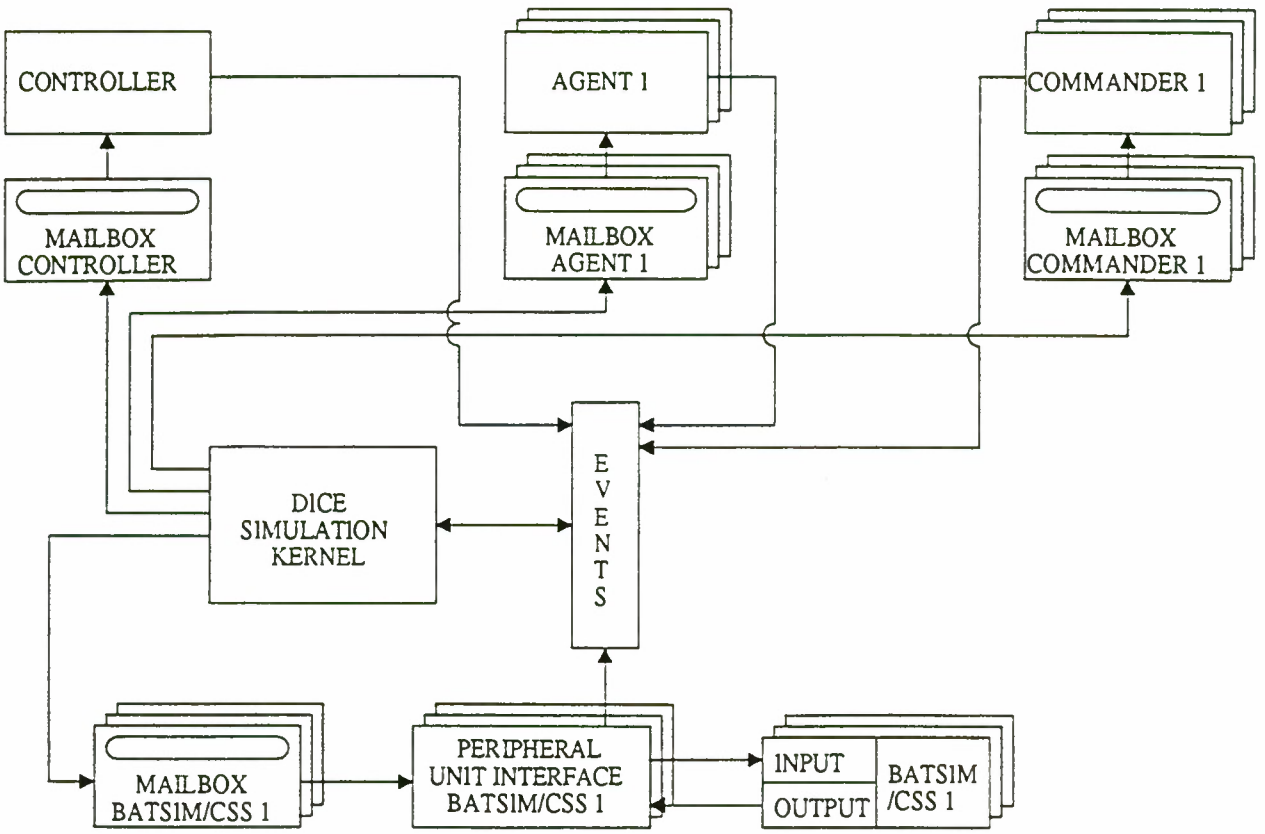


Figure 2: DICE simulation kernel and communication

4.2 Simulation control

The simulation control function refers to the ability to define and represent a scenario, simulate it and carry out analysis on that simulated scenario prior to, during and after execution.

One feature of the scenario generator suite of tools that has been developed is a general-purpose graphics-based drawing environment *ScenGenDraw* which enables the definition of an organisational structure of network entities. The general-purpose nature of this tool is such that a defined network can be used for run-time display and also as a front end to other layers of the scenario generator environment. ScenGenDraw can also be used for graphical construction of the rules describing artificial agent behaviour. Features of the scenario generator include the ability to specify the characteristics of nodes and links; allocate human commanders as players; specify and assign artificial agents; and establish the links with required peripheral units.

The simulation controller or analysts can program messages to be received by nodes, at given times in the simulation, before the simulation is started. Such events are referred to as independent external stimuli since they are pre-scheduled and hence independent of activities that occur whilst the simulation is running (Davies 1993). Recent work allows the controller to communicate with all players (real or artificial) and to inject stimuli during run-time into the simulation. The controller can secretly pretend to be any one player and send messages to others if he wishes to influence the simulation execution in that way. The controller can pause, advance and resume the simulation as needed.

Other developments include the ability to specify and inspect the standard messages that apply in a scenario. This is important to the simulation analyst that assists with setting up the scenario to be addressed. An analyst can inspect characteristics of ADF ADFORMS messages that might apply in a scenario but, more importantly, this facility gives the ability to define custom ADFORMS-like messages. This is important since there will be a need to emulate many forms of communication (including

telephone, fax etc) for which ADF ADFORMS messages might not exist.

4.3 Human commanders

The main function associated with the human commanders that form players in the simulation, is the ability to receive, create and submit messages such that communication with the remainder of the simulation is achieved. Preliminary Graphical User Interfaces (GUI) have been established such that a basic capability to create ADFORMS messages is enabled; however, the eventual intention is to utilise original or tailored versions of operational software that provide such a capability to real-world commanders. Hence, CSS will be used, as required, by human players in the DICE simulation. It is considered useful if human commanders have access to some form of analysis capability and an example of which is the ability to retrieve summary information on their performance and impact after simulation execution. Such a capability will be achieved through the run-time, pre- and post-simulation analysis facility.

4.4 Artificial agents

The main function associated with artificial agents in the DICE simulation is similar to that of the human players. The artificial agents need to be able to communicate with the human commanders in a common form, namely through a chosen textual formatted message system. The artificial agents need to be able to interrogate, recognise and react to such messages and need to be adequately realistic representations of the real-world agents that they portray. The underlying assumptions associated with such agents need to be easily conveyable to an analyst or a military domain expert who may be assessing the credibility of the artificial representation.

Initial developments of artificial agents have concentrated on simple and rule-based representations but it is expected that they will become more sophisticated as the full capabilities of the artificial intelligence field are investigated. The form of the rules needs to be such that an artificial agent can be created from building blocks that describe basic functions or roles that are not peculiar to a given agent in a given scenario. Research has been carried out into the suitability and use of Petri nets in this area and this is the current technique that is being employed in representing artificial agents in the DICE simulation (Bowden et. al. 1995). Each agent, then, is a data-driven Petri net simulation and development of a GUI environment for the description and implementation of such agents has commenced based on the SceneGenDraw software. This environment will also allow access to an associated agent explanation and analysis capability that has been developed using the declarative programming language Prolog (Bowden et. al. 1995). Seamless integration of artificial and real players, machine learning, and other areas of artificial intelligence are all of relevance here.

4.5 Run-time, pre- and post-simulation analysis

This facility will allow interrogation and retrieval of database information prior to, during and after execution of a given scenario. Such a capability will facilitate establishment of a scenario; run-time communication; and post-simulation analysis and review. Post-simulation analysis includes inspection of C3I system characteristics such as bottlenecks; command and information flow parameters; and effectiveness measures. The history and review function must also provide the ability to select and replay aspects of the simulation including analysis of the impact of key commands or decisions made by artificial or human players. Configuration of the simulation kernel about an Ingres database will facilitate establishment of a recording and analysis system.

4.6 Peripheral units (battle simulations, war games, CSS)

Tactical-level (eg battlefield) simulations are used to represent activities that are external, or at a lower level, to the main C3I simulation and help portray the overall military mission concerned. Such representations allow the impact of C3I aspects to be gauged and hence their effectiveness determined. As mentioned earlier, interfacing of peripheral units to the main DICE simulation is achieved through PUI. A standard shell architecture has been developed for PUI which includes the concept of *tassels* which relates to the PUI having a number of arms emanating from it, the loose ends of which need to be tied down when that PUI (and hence the associated peripheral unit itself) is employed in building a scenario. The tassels signify the different categories of information that a peripheral unit can supply to other nodes in the simulation. Tying of the tassels specifies what entities in the simulation require what information from the peripheral unit.

Peripheral units can broadcast all information to all nodes if required, which may have particular benefits (Miller 1992).

Although some CSS may be regarded as stand-alone or off-line, generally the interfacing problems that apply to battlefield simulations apply here also. CSS are generally driven by real-world data which, in the DICE environment, is simulated.

Investigations have been made into the feasibility of interfacing selected tactical-level simulations and CSS, for scenarios of immediate interest, with the DICE simulation. An in-house developed air picture simulation has been interfaced to and used to demonstrate many of the concepts of the DICE project including MOE evaluation. Current efforts in this area include investigation of DIS procedures to enable interfacing with units that are geographically distributed.

5. Conclusions

The DICE simulation environment will permit a thorough examination of current and proposed C3I-related procedures and technologies within the framework of simulated military missions. The aims and current status of the DICE program have been reported in this paper. Clearly, there are a number of aspects of the program which are yet to be finalised. However, an evolutionary approach has been adopted and at each stage a functioning facility will be available. The DICE activity includes future research into a number of areas related to military C3I. Appropriate measures of effectiveness will need to be defined and battlefield and other models will be used to assess these measures by reason of the impact of C3I aspects on overall military missions.

A basic environment is currently established which relies heavily upon a tightly orchestrated scenario utilising a limited set of standard messages. Such orchestration will be progressively relaxed as the sophistication and capabilities of the artificial agents, for example, increases. One of the main current activities is to establish a library of peripheral units that can be called upon as required and used to address specific ADF requests.

6. Acknowledgements

The authors would like to thank Mr Jonas Plumecocq for his assistance in simulation kernel design and development.

7. References

Australian Government Publishing Service, "Defending Australia", Defence White Paper, Canberra, 1994

Bowden, F.D.J., Davies, M., Dunn,J.M., "Representing role-based agents using coloured Petri nets", Proc. 5th CGF&BR Conf., Orlando, Florida, May 1995

Davies,M., 'Strategic command, control, communication and intelligence (C^3I) simulation activities', Int. Cong. on Modelling and Simulation, Perth, Australia, 6-10 December 1993

Fogg, D.A.B., "Interactive C3I Simulation with Additional Synthetic Decision Makers", Proceedings XXXII US Army Operations Research Symposium, Fort Lee, Virginia, October 1993

Miller,D.C., 'Interoperability issues for distributed interactive simulation', Proc. Summer Computer Simulation Conf., 1992, pp1015-1018

8. Authors' Biographies

Mike Davies is a research scientist with the Defence Science and Technology Organisation in Australia. He has a BSc(Hon) in applied mathematics and a PhD in mathematical modelling. His research interests are in the areas of military operation research, mathematical modelling and computer simulation.
(Tel: +61 8 259 6613; Fax: +61 8 259 6781;
EMail: Michael.Davies@dsto.defence.gov.au)

Carsten Gabrisch is a professional officer with the Defence Science and Technology Organisation in Australia. He has a BSc in applied mathematics. His interests are in the areas of computer simulation over a distributed network, relational databases and mathematical modelling.
(Tel: +61 8 259 5314; Fax: +61 8 259 6781;
EMail: Carsten.Gabrisch@dsto.defence.gov.au)

Integrated Eagle/BDS-D: A Status Report

Robert W. Franceschini
Institute for Simulation and Training
3280 Progress Drive, Orlando FL 32826-0544
rfrances@ist.ucf.edu

1. Abstract

Integrated Eagle/BDS-D was the first published constructive+virtual linkage. The project demonstrated the concept of linking the Eagle constructive simulation with DIS/SIMNET using the Institute for Simulation and Training's Computer Generated Forces Testbed (IST CGF Testbed). Ongoing work on this project has continued to push into new territory by expanding the capabilities of the IST CGF Testbed to accommodate envisioned uses of this linkage and by studying and proposing DIS standards for linking constructive and virtual simulations. This paper reports the latest work on the Integrated Eagle/BDS-D project.

2. Background

2.1 Constructive Simulations

Many constructive simulations, or wargames, represent aggregate units in combat scenarios using discrete time and space representations. Such simulations have historically been used for analysis of tactics or weapons effectiveness in large scale battles (for example, in corps or division level battles). Some examples of constructive simulations include BBS, CBS, and Eagle.

There are many simulations that are called constructive simulations. Some are capable of representing individual vehicle platforms. For our purposes, we are specifically interested in constructive simulations which do not represent individual vehicles; the smallest representable units on such systems are typically battalions or companies. However, many of the same techniques used in dealing with these aggregate constructive simulations are applicable to vehicle level constructive simulations.

2.2 Virtual Simulations

Virtual simulations represent individual vehicle platforms, dismounted infantry fireteams, or individual human combatants as simulation entities using continuous time and space. Such simulations are traditionally used for small unit training in cooperative maneuvers and close combat. Some examples of virtual simulations include DIS and SIMNET.

2.3 Constructive+Virtual Simulations

A constructive+virtual simulation links a constructive simulation with a virtual simulation. For a tutorial on constructive+virtual simulations, see Franceschini (1995b).

Many technical problems must be overcome to successfully link constructive and virtual simulations. Time and space are handled differently in constructive and virtual simulations. Constructive simulations typically are time-stepped, and the amount of time required to process the actions during a time step is usually not related to the simulation time elapsed during the time step. However, virtual simulations are expected to be real-time, so that the amount of time required to process actions approximates the simulation time elapsed. Constructive simulations usually have an abstract terrain representation; they do not need a detailed polygonal or elevation post database since their smallest units are company-sized. However, since virtual simulations model individual entities, they require a much higher level of detail in their terrain representation. These two differences between constructive and virtual simulations are at the root of most of the difficulties in linking these types of simulations.

We will distinguish between *units* and *entities* in this paper. A unit is an aggregate controlled by the constructive simulation. An entity is an individual vehicle platform, infantry fireteam, or individual combatant controlled by the virtual simulation.

3. The Integrated Eagle/BDS-D Project

3.1 History of Integrated Eagle/BDS-D

The Integrated Eagle/BDS-D project began in 1992. During the first phase of the project, several interactions were implemented across the boundary between constructive and virtual simulations: disaggregation, aggregation, transfer of operations orders and CGF operator intent, and indirect fire (Karr 1992) (Karr 1993). In addition, IST addressed the problem of instantiating entities during disaggregation so that both the formation template of the unit and the

terrain around the entities are considered (Franceschini 1992).

In the next phase, the Integrated Eagle/BDS-D project team generalized and extended the system (Karr 1994). The concept of disaggregation was generalized to include disaggregations for units of any size (Karr 1994). The possible representations for units in the virtual simulation were extended (the project introduced the concept of pseudo-disaggregation) (Root 1994). Indirect fire support was extended to allow indirect fire from disaggregated artillery batteries at units in the constructive simulation (Karr 1994). Other results can be found in (Karr 1994).

3.2 New Work

Since the Fourth CGF Conference, several improvements have been made in the Integrated Eagle/BDS-D system. The focus of much of this work has been in preparing the Integrated Eagle/BDS-D system for use in the Aviation Testbed at Fort Rucker AL. Since aviation scenarios have different characteristics than ground scenarios, certain modifications were required to have a meaningful simulation. The underlying IST CGF Testbed's capacity has been increased, the system has been modified to allow its operation in DIS as well as SIMNET, and new helicopter behaviors have been developed.

New systems based on the ideas pioneered in the Integrated Eagle/BDS-D project are being constructed; IST has participated in the design of these systems by giving in-depth discussions of the details of the Integrated Eagle/BDS-D system and by commenting on strawman designs. The Integrated Eagle/BDS-D project team has also begun an effort to survey constructive+virtual systems and propose new DIS protocol standard PDUs intended for linking constructive simulations to DIS.

4. System Enhancements

4.1 Increasing the Capacity of the IST CGF Testbed

One use for constructive+virtual simulations is to extend the virtual battlefield by allowing small scale virtual exercises to be played in the context of a large scale constructive exercise. However, the small scale exercise played in the virtual simulation must be large enough to have meaning in the overall scenario. For the air scenario to be used at Fort Rucker, IST determined that the IST CGF Testbed needed to support many more disaggregated vehicles than its original implementation allowed.

The IST CGF Testbed runs on IBM-compatible personal computers. Upon analysis of the IST CGF Testbed, it was determined that the factor limiting the number of vehicles supported by the system was the amount of random access memory (RAM) available to the program. Due to its implementation, the IST CGF Testbed was limited to 640 kilobytes of RAM. IST increased this limit by switching to a compiler that supports a 32-bit flat-memory model. This increased the number of vehicles the IST CGF Testbed could support by nearly 300%. At this point, we believe that the capacity of the IST CGF Testbed is now limited only by the computational power of the host computer's processor, not on memory limitations.

More information on the increase in capacity of the IST CGF Testbed can be found in a companion paper (Schricker 1995a). This paper gives more details on the task and presents the results of performance analyses of the IST CGF Testbed.

4.2 DIS Compliance

A long term goal of the Integrated Eagle/BDS-D project has been to transition the system to DIS compliance. During the last year, the necessary modifications were made to the system to allow it to participate in exercises using the DIS 2.0.3 protocol standard. Because of the design of the system, the pieces affected were the IST CGF Testbed, including the Eagle CGF Manager, and the SIU (See (Karr 1994) for a discussion of the architecture of the Integrated Eagle/BDS-D system).

Converting the IST CGF Testbed to the DIS protocol was a simple matter. The IST CGF Testbed's design allows different network protocols to be swapped easily (as a link-time option). Since a module containing the DIS protocol was already implemented at IST, the conversion process consisted of building the executable code and performing some small testing and debugging tasks (no new code was required).

Converting the Eagle CGF Manager and the SIU to the DIS protocol was a bit more involved. The Interoperability Protocol used for communications between the Eagle CGF Manager and the SIU was switched from broadcast to point to point. Application specific point to point communications are allowed by the DIS protocol standard.

The most interesting DIS protocol issue that was handled in this conversion was the method of representing aggregate units in DIS. In SIMNET, Vehicle Appearance PDUs are capable of representing aggregate units (since echelon information is encoded in the PDUs). Therefore, the original SIMNET implementation of the system broadcast Vehicle Appearance PDUs on the network to represent the position and composition of aggregate units under Eagle's control. However, the Entity State PDU in DIS does not support echelon information. Further, the Aggregate Protocol in DIS was experimental at the time of this conversion, and was judged to be insufficient for the purposes of the Integrated Eagle/BDS-D system. To solve this problem, the Integrated Eagle/BDS-D system used an application specific message type to transmit aggregate unit information to the IST CGF Testbed components on the DIS network. As the DIS Aggregate Protocol evolves, we plan to use it for this purpose.

4.3 Helicopter Behaviors

The IST CGF Testbed was developed concentrating almost exclusively on land combat (Smith 1992). As a result, behaviors for helicopter platforms are less sophisticated than behaviors for land platforms. Examples of these limitations were quickly discovered when the Integrated Eagle/BDS-D system transitioned to an air scenario. In early interactions with US Army soldiers in manned Apache simulators, the CGF helicopters were unable to fly realistically close to the terrain at high speeds. The IST CGF Testbed was enhanced to allow the CGF helicopters to fly more realistically. More information about the algorithm developed and the results obtained from this work can be found in a companion paper (Schricker 1995b).

5. Installations and Demonstrations of the System

5.1 Fort Rucker's Aviation Testbed

The Integrated Eagle/BDS-D system was installed for the first time at Fort Rucker's Aviation Testbed during July 1994. A first-cut scenario was demonstrated for MG Robinson of Fort Rucker. This scenario demonstrated the capabilities of the Integrated Eagle/BDS-D system, including the ability to include manned helicopter simulators as part of disaggregated units. Many of the system enhancements described in Section 4 came as a result of experiences with this first installation.

The latest version of the Integrated Eagle/BDS-D system was installed at Fort Rucker during February 1995. This version of the software included the capacity and helicopter behavior enhancements discussed in section 4 (the Aviation Testbed is a SIMNET facility, so the Integrated Eagle/BDS-D system that was installed used the SIMNET protocol). This new system was demonstrated for the Assistant Secretary of Defense and for MG Adams.

5.2 I/ITSEC

The Integrated Eagle/BDS-D system participated in the DIS Interoperability Demonstration at the Interservice/Industry Training Systems and Education Conference (I/ITSEC) in late 1994. The most significant technical results of this demonstration were that the DIS version of the Integrated Eagle/BDS-D system (described in Section 4.2) passed the rigorous DIS compliance testing required of all participants and that the capacity increase of the IST CGF Testbed (described in Section 4.1) allowed the system to handle a simulation load of approximately 30 disaggregated entities with a network load of approximately 100 total entities.

6. Extensions of Integrated Eagle/BDS-D

6.1 New Constructive+Virtual System

A testament to the success of the Integrated Eagle/BDS-D project is that the general architecture developed on this project is the basis for another constructive+virtual simulation: Corps Level Computer Generated Forces. The Corps Level Computer Generated Forces project links Eagle with DIS using ModSAF as the CGF system; it is described in (Calder 1995a) and (Calder 1995b).

6.2 Survey of Constructive+Virtual Simulations

IST has begun an effort to learn about and catalog existing constructive+virtual simulations. The goals of this work are to provide the simulation community with a survey of the work that has been done and to draw some generalizations that can be used to standardize the process of linking constructive and virtual simulations. The initial results of the survey are available in a companion paper (Kraus 1995); IST is still collecting information about other constructive+virtual linkages to be included in future versions of that paper.

As part of this work, IST has begun examining the existing constructive+virtual simulations and the DIS protocol in an effort to determine whether new PDUs should be added to the DIS protocol to standardize the

integration of constructive simulations into DIS (Franceschini 1995a).

6.4 Spreading Disaggregation

Finally, the Integrated Eagle/BDS-D project team has considered one problem that is common in constructive+virtual linkages: the problem of spreading disaggregation. Disaggregation can be triggered in many different ways (see (Franceschini 1995b) for examples of disaggregation triggering mechanisms); ideally, the constructive+virtual simulation should trigger disaggregation automatically and dynamically during a scenario. However, currently proposed automatic and dynamic disaggregation triggers suffer from the fact that, in certain conditions, they can cause disaggregation of an undesirably large number of units; this is called spreading disaggregation. More information about spreading disaggregation can be found in a companion paper (Petty 1995).

7. Conclusions

After demonstrating the concept of linking constructive and virtual simulations, the Integrated Eagle/BDS-D project has pushed forward in two directions. First, the project is preparing for the use of constructive+virtual simulations in training and analysis activities. Second, the project seeks to standardize the methodologies developed for linking constructive and virtual simulations to make it easier for future systems to be developed.

Constructive+virtual simulations are viewed as a solution to a current limitation of DIS exercises: the network of simulators is incapable of supporting very large scale exercises. They are also viewed as a new and powerful tool for conducting analytic studies, and as a way to bring legacy systems into the DIS environment. As a result, many new constructive+virtual simulations are being built. Integrated Eagle/BDS-D is building a path for designers of those systems to follow.

8. Acknowledgment

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command (STRICOM) and by the US Army TRADOC Analysis Center as part of the Integrated Eagle/BDS-D project, contract number N61339-92-K-0002. The survey of constructive+virtual linkages was sponsored by the US Army STRICOM as part of the Signal Intelligence/Electronic Warfare project, contract number N61339-93-C-0091. That support is gratefully acknowledged.

9. References

Calder, R.B., Peacock, J.C., Panagos, J., and Johnson, T.E. (1995a). "Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Calder, R.B., Peacock, J.C., Wise, B., Stanzione, T., Chamberlain, F., and Panagos, J. (1995b). "Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Franceschini, R.W. (1992). "Intelligent Placement of Disaggregated Entities", *Proceedings of the 1992 Southeastern Simulation Conference*, Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 20-27.

Franceschini, R.W. and Petty, M.D. (1995a). "Status Report on the Development of PDUs to Support Constructive+Virtual Linkages", *Proceedings of the 12th Workshop on Standards for the Interoperability of Distributed Simulations*, Institute for Simulation and Training, Orlando FL, March 13-17 1995.

Franceschini, R.W. and Petty, M.D. (1995b). "Linking Constructive and Virtual Simulation in DIS", *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing & Control and Dual-Use Photonics*, Orlando FL, April 17-21 1995.

Karr, C.R., Franceschini, R.W., Perumalla, K.R.S., and Petty, M.D. (1992). "Integrating Battlefield Simulations of Different Granularity", *Proceedings of the 1992 Southeastern Simulation Conference*, Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 48-55.

Karr, C.R., Franceschini, R.W., Perumalla, K.R.S., and Petty, M.D. (1993). "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, March 17-19 1993, pp. 231-239.

Karr, C.R. and Root, E.D. (1994). "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on*

Computer Generated Forces and Behavioral Representation, Institute for Simulation and Training, Orlando FL, May 4-6 1994, pp. 425-435.

Kraus, M.K., Stober, D.R., Foss, W.F., Franceschini, R.W., and Petty, M.D. (1995). "Survey of Constructive+Virtual Models", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Petty, M.D. and Franceschini, R.W. (1995). "Disaggregation Overload and Spreading Disaggregation in Constructive+Virtual Linkages", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Root, E.D. and Karr, C.R. (1994). "Displaying Aggregate Units in a Virtual Environment", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 4-6 1994, pp. 497-502.

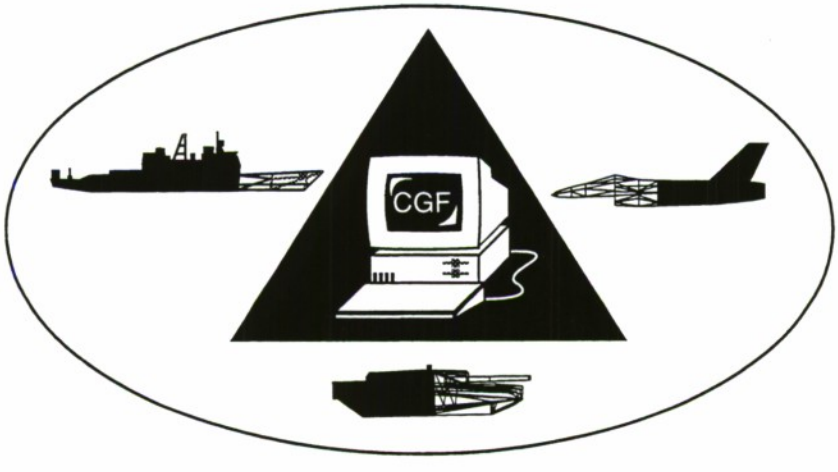
Schricker, S.A., Tolley, T.R., and Franceschini, R.W. (1995a). "Benchmarking and Optimization of the IST CGF Testbed", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Schricker, S.A., Franceschini, R.W., Petty, M.D., and Tolley, T.R. (1995b). "Terrain Avoidance for CGF Helicopters", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Smith, S.H., Karr, C.R., Petty, M.D., Franceschini, R.W., Wood, D.W., Watkins, J.E., and Campbell, C.E. (1992). "The IST Semi-Automated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, February 28 1992.

10. Author's Biography

Robert W. Franceschini is a Principal Investigator at the Institute for Simulation and Training. He currently leads the Integrated Eagle/BDS-D project at IST. Mr. Franceschini earned a B.S. in Computer Science from the University of Central Florida; he is currently pursuing an M.S. in Computer Science at UCF. His research interests are in simulation, graph theory, and computational geometry.


CGF

Simulated Intelligent Forces For Air: The Soar/IFOR Project 1995

John E. Laird,[1] W. Lewis Johnson,[2] Randolph M. Jones,[1] Frank Koss,[1] Jill F. Lehman,[3]
Paul E. Nielsen,[1] Paul S. Rosenbloom,[2] Robert Rubinoff,[3] Karl Schwamb,[2]
Milind Tambe,[2] Julie Van Dyke,[3] Michael van Lent,[1] and Robert E. Wray, III[1]

[1]Artificial Intelligence Laboratory
University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110
laird@umich.edu

[2]Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
rosenbloom@isi.edu

[3]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
jef@cs.cmu.edu

1 Abstract

For the last three years, the Soar/IFOR group has been developing intelligent forces for distributed interactive simulation environments. Since early 1994, our efforts have been focused on developing computer generated forces for air missions including both fixed wing and rotary wing aircraft. This paper reviews the current state of the Soar/IFOR project and discusses the results of a preliminary trial of our agents in STOW-E, a precursor to STOW-97.

2 Introduction

The goal of the Soar/IFOR project is to develop human-like synthetic agents for populating interactive distributed simulation environments. In contrast to the standard semi-automated forces (SAF) approach, where it is assumed that some higher-level authority, such as a human or a computerized command force (CFOR), will be responsible for all decisions requiring judgement, our approach is to endow all entities with knowledge and decision making abilities similar to those found in humans performing similar tasks. Our hypothesis, confirmed in part by our participation in a large scale simulated exercise called STOW-E, is that building intelligent forces provides a payoff in terms of increasing the fidelity of the agents' behavior, while decreasing the complexity of commanding the agents.

From 1992 through early 1994, our efforts were focussed on research and development for beyond visual range air-to-air combat leading to the creation of TacAir-Soar [Jones *et al.*, 1993; Rosenbloom *et al.*, 1994; Tambe *et al.*, 1995a]. In early 1994, we broadened our horizon significantly, and we are now working on developing automated synthetic pilots for the majority of air missions flown in the U.S. military. The various missions include air-to-air (defensive combat air patrols, sweeps), air-to-ground (close air support, interdiction, strategic attack), air-to-surface, rotary wing (anti-armor), as well as some support missions (refueling, resupply, etc.). We are also developing additional agents, such as air and ground controllers, that communicate with the agents flying in planes and helicopters during their missions. We will refer to all the agents being developed by the Soar/IFOR project as Air-IFOR agents, while TacAir-Soar refers to the agents that fly tactical fixed wing aircraft.

During the last year, we have made progress on many of these missions, and in this paper we will review all aspects of the existing Soar/IFOR agents, including: the interaction between Air-IFOR agents and DIS, the design of Air-IFOR agents, their capabilities, the interactions between multiple Air-IFOR agents, and the participation of Air-IFOR agents in STOW-E.

3 Interaction with DIS

Since the inception of the Soar/IFOR project, our goal has been to create an abstract interface layer between Air-IFOR agents and the underlying simulation (DIS) environment. We call this

the "virtual cockpit" abstraction, meaning that Air-IFOR agents should have an interface that supports the types of interactions a pilot has in the cockpit of a plane or helicopter [Schwamb *et al.*, 1994]. Thus, Air-IFOR agents are isolated from the details of the underlying simulation environment, network protocol, plane dynamics, sensor simulation, etc. Currently, we use ModSAF [Calder *et al.*, 1993] as the underlying software which provides connectivity to the DIS environment as well as simulations of the vehicle dynamics, sensors, weapons, and communication (radio) systems. To support the virtual cockpit, we have added C code, which defines a Soar/ModSAF Interface (SMI)[Schwamb *et al.*, 1994]. The SMI makes all of the appropriate calls to the underlying ModSAF functions so that Air-IFOR agents get access to the appropriate sensor and weapons systems. The SMI does not use ModSAF tasks or taskframes, but instead relies on lower level functions which gives Air-IFOR agents finer-grain control of their own behavior.

Air-IFOR agents are built within the Soar architecture [Laird *et al.*, 1987; Laird and Rosenbloom, 1994; Rosenbloom *et al.*, 1991; Rosenbloom *et al.*, 1993]. Soar, the SMI, and ModSAF are integrated (within the same Unix process) so that each Soar/IFOR agent gets "ticked" during the simulation cycle. Using this arrangement, we can run multiple, independent agents on a single Unix workstation, as well as having agents on many different machines — although a single agent is not distributed across multiple machines. Air-IFOR agents do not share data except through explicit communication using simulated radios.

As part of building the SMI, we have extended the standard suite of ModSAF sensors and weapons, adding such devices as a CCIP (continuously computed impact point) which displays where a bomb will hit if released, a waypoint computer which displays the appropriate heading to fly to the next waypoint in a flight plan, air-to-surface missiles (such as the Exocet), and a primitive form of precision-guided munitions.

One result of our development has been the recognition that the closer we model the types of information available to humans, not at the level of visual perception, but instead at the level of symbolic data, the easier it is to model the behavior of the humans. For example, we discovered that creating a waypoint computer and the CCIP greatly reduced the reasoning required by the Soar agents because they no longer had to respond to every change in their position relative to a waypoint or target. Instead they could respond to the changes in the heading suggested by the waypoint computer or CCIP.

A problem we foresee in the future is the management of many Soar/IFOR agents during a protracted exercise. The problem is not in terms of command and control (covered in the section on multiagent interactions), but is in terms of managing the creation, reuse, and destruction of Air-IFOR agents on many different workstations. To this end, as well as to support cleaner interfaces to Soar agents, we have integrated Soar with Tcl [Ousterhout, 1994], a scripting language, that will help support agent management across many machines.

4 Agent Design

The overall design of Air-IFOR agents has not changed significantly over the last year, although it has been refined and augmented with new tools. Nor have the basic requirements of Air-IFOR agents changed. They continue to be the following:

1. Encode large bodies of knowledge about relevant aspects of the world, including tactics, doctrine, sensors, weapons, etc.
2. React quickly to the environment, such as the behavior of enemy planes, communications from other friendly agents, and changes in terrain being traversed.
3. Determine the tactically relevant features of a complex, dynamic environment.
4. Coordinate behavior with other agents.
5. Use minimal computational resources.
6. Deliberately plan aspects of missions not specified in orders.

4.1 Method and Approach

All of the Soar/IFOR agents are developed within the Soar architecture. Soar has its roots in early AI symbolic systems such as LT [Newell and Simon, 1956], and GPS [Ernst and Newell, 1969], as well as rule-based systems, such as OPS5 [Forgy, 1982]. Soar supports the above requirements by providing two integrated levels of computation: deliberate, sequential operators within problem spaces, and automatic parallel rules. In terms of the tasks that have to be performed by Air-IFOR agents, it is easiest to think in terms of the first level, operators. We make the claim that sequences of deliberate operators are the most appropriate way to model the second to second behavior of a pilot (or any human for that matter). Example operators include flying a mission, picking a control point to fly to, intercepting a bandit, entering a waypoint into the plane's waypoint computer, deciding which missile to fire, physically selecting that missile, pushing the fire button, and so on. Some of these are purely mental operators, such as deciding which missile to select, while others include physical actions. Many

of these operators cannot be performed directly as a single act, but instead must be decomposed into subgoals where finer-grain operators are selected and applied. For example, the act of intercepting a bandit is decomposed into many different operators, such as achieving proximity, employing weapons, and so forth.

Thus, Soar organizes the doctrine and tactics of flying missions in planes and helicopters in terms of hierarchies of operators. For a given operator that the agent is trying to pursue, such as an intercept, the operators used to achieve it are grouped in terms of *problem spaces.* They are called problem spaces because their constituent operators determine the space of problems that can be solved. Operators can be shared among more than one problem space. For example setting the waypoint computer is used in flying routes, as well as flying BARCAPs. Other, so-called, *floating operators,* are available in every active problem space. Floating operators such as operators that detect changes in a bogey's activity, are very sensitive to changes to the environment and usually need to be selected soon after they become relevant. More generally, the hierarchical and floating operators can be seen as at opposite ends of two dimensions: sensitivity to the agent's current goals, and sensitivity to the current situation. All operators must be sensitive to both concerns, but floating operators emphasize reacting to the current situation (within the context of the current goals), while hierarchical operators emphasize responding to the current goals (within the context of the current situation).

Within a subgoal, local situational information is held in the subgoal's *state.* Each subgoal has access to all of the state information in its supergoals, and the state of the top goal contains all the data used to fly a mission, including all sensor data, the agent's interpretation of the current situation, a description of the current mission, data on other agents, etc.

The hierarchical operator structure provides the necessary framework for encoding knowledge and organizing the behavior of Air-IFOR agents; however, it alone is insufficient to provide flexibility and reactivity. What is needed is the ability to dynamically propose, select, and apply the operators that are appropriate for the current situation. This is done in Soar through its underlying rule-base system, which directly implements the selection, application, and termination of operators described above. Thus, there are rules which test the current situation and propose operators, rules which compare proposed operators and suggest *preferences* between operators, rules which test that an operator has been selected and then performs some aspect of the operator, and rules that test that all aspect of an operator have been completed, and signal that the operator is finished. The actual selection of operators is not done directly by individual rules, but by a *decision procedure*, which selects an operator based on all relevant preferences.

Most rule-based systems use a conflict-resolution scheme to select a single rule to fire on each cycle. However, rules from these systems map more directly onto Soar's operators, which are the locus of deliberate activity in Soar, and where selection is controlled by preferences and the decision procedure. Soar's rules are more like an associative memory, where the information in actions of rules is recalled whenever the conditions of the rules match. Thus to retrieve all information relevant to the current situation, the basic cycle is to fire all rules that match the current situation, and continue firing until quiescence. During this rule firing phase, rules to implement the current operator are firing, as well as rules proposing new operators. At quiescence, assuming the current operator is finished, a decision is made to select a new operator based on the available preferences, and the cycle begins again. If the current operator cannot be finished, possibly because it requires problem solving in a subgoal, a subgoal will be created automatically, and then rules sensitive to the subgoal will fire to suggest appropriate operators. When a rule detects that the original operator is finally complete (or should be abandoned), it will fire and cause a new operator to be selected and the immediate subgoal (and any additional subgoals) will be automatically removed. Soar is integrated with ModSAF so that one decision is made for each agent during each clock tick of the simulation, and thus 2 to 15 decisions are made in each Air-IFOR agent each second.

4.2 Infrastructure

In maintaining a rule-based system, the rules must be organized so that it is easy to find rules, not only by their name, but also by their role in producing behavior. For the Soar/IFOR agents, we have mapped the hierarchical structure of the operators onto the hierarchical structure of the Unix file system. Thus, each goal (or subgoal) has its own directory, and within that directory there are files for each of the operators, plus a file for loading in those operator files. For cases where rules are not shared across agents, we have a dynamic load facility that loads only the subset of the code that is relevant to the current agent's vehicle and mission.

Our lowest-level documentation of the problem space, operators, and rules is also organized in the same hierarchical file structure with direct links from the documentation to the code [Koss and Lehman, 1994]. A higher level of documentation,

using the terminology and structure of our domain experts, links into the problem space documentation to currently support a limited form of validation. All of our documentation is in HTML and it can be accessed through viewers such as Mosaic and Netscape.

To support the creation of the code and documentation with our conventions, we have created the Soar Development Environment (SDE) [Hucka and Laird, 1995], which is an extension to Emacs. SDE has a template language that can be used to automatically generate all of the necessary directories, code, and documentation files when new operators are created. SDE also provides many features to aid in debugging, such as automatic finding of files in which rules are stored, point and click commands for common functions, and general search facilities for the rules.

4.3 Current Status and Lessons Learned

The current Air-IFOR agents have a combined total of approximately 320 operators, with a total of 3,100 rules. Individual agents have between 1,130 and 2,550 rules depending on their missions. These counts do not include our natural language or debriefing systems, which by themselves have substantial numbers of rules.

One of the challenges in building the agents has been to maintain the computational efficiency of the system as we add new capabilities. The problem is not that Soar slows down as the sheer number of rules increase (research indicates that Air-IFOR agents may be able to grow to even a million rules without this being an issue [Doorenbos, 1994]), but instead the problem is that it is easy to write rules that fire every time some input data changes (such as when the current position of the plane changes). As a result, we closely monitor rule firings in order to identify costly rules, and then attempt to rewrite them in order to decrease their cost. In a few cases, we have discovered that by removing a computation from Soar that is done in the cockpit for a pilot, such as with the waypoint computer and the CCIP, we have been able to drastically reduce the computational overhead in Soar.

During agent development, we are able to run 6-10 agents on a single 150MHz 4400 SGI Indy. However, one of the lessons we learned from STOW-E is that we are limited to around 4 agents when there are large numbers of entities on the network. This is because of overhead in both ModSAF and the Soar agents that results from the processing of large numbers of entities. In response, we expect to put more emphasis on focusing attention on only the most important entities at all levels of processing, as well as to continue research on efficient matching of rule-based systems [Acharya and Tambe, 1993; Kim and Rosenbloom, 1993].

5 Agent Capabilities

Although Soar provides the basic architecture for building Air-IFOR agents, our agents are more than a large collection of rules that directly encode doctrine and tactics. They must also have a many cognitive capabilities, some of which are directly related to military flying such as following a flight plan, situational awareness, planning attacks, employing weapons, and managing fuel, while others are more general cognitive capabilities, such as communicating with other agents, modeling the behavior of other agents, being able to explain the agent's behavior, and using general problem solving strategies.

To date, we have discovered that although these general cognitive capabilities are important, we have been able to build viable agents by concentrating on those capabilities directly related to performing our agents' missions. Thus, we have developed and incorporated capabilities for following flight plans, planning attacks, employing weapons, situational awareness, managing fuel, and so on. All of these are the building blocks for various missions. There are also many capabilities dealing with coordinating behavior among multiple agents, which are discussed in the section on multiagent interactions. These capabilities are all implemented as operators that have complex subgoals. For example, following a flight plan involves many operators including flying routes (of which there are different types depending on the aircraft), performing various activities at waypoints (such as communicating with control agents or determining if a plane should delay at the point so that it arrives on target at the appropriate time), selecting the next route, and processing any changes the agent might receive to its mission. We expect these capabilities to be reused on future missions, possibly with modification as new variants are required.

We expect that the more general cognitive capabilities will become necessary as we try to create agents which are more autonomous, and thus able to handle novel situations on their own. To that end, we are pursuing research in the following areas:

1. Natural language processing: Even with the advent of the Command and Control Simulation Interface Language (CCSIL) [Salisbury, 1995], we will someday want Air-IFOR agents to directly interact with humans. Air-IFOR agents will need to understand and generate natural language, with one of the challenges being to integrate the processing of language

with all of the other agents' tasks [Lehman *et al.*, 1995].

2. Behavior explanation: As the complexity of Air-IFOR agents grow, it is necessary for each of them to be able to explain its own behavior and internal reasoning. What action did it take, why did it take that action, why did it interpret the situation in the way it did, and what were other options? We have been actively pursuing these issues in the Debrief system, which is a set of Soar rules that when included in an agent before a run, allows the agent to be debriefed after flying a mission [Johnson, 1994].
3. Agent modeling: In order to interpret the actions of other agents, Air-IFOR agents must have some understanding of what the other agents are thinking. This is currently done in very specialized and context specific ways in Air-IFOR agents. However, as we start to explore complex behavior, it will be necessary for Air-IFOR agents to create general internal models of what other agents are thinking about the current situation. For example, deceptive maneuvers involve generating behaviors with the goal of leading an opponent to incorrectly guess what your intent and action really is. We can currently encode "deceptive" maneuvers in Air-IFOR agents; however, for the agent itself to derive an appropriate deceptive maneuver in novel situations requires the ability to internally model some of the thought processes of other agents, a problem we are actively pursuing [Tambe and Rosenbloom, 1995].
4. General Problem Solving and Planning: Our current agents have all the necessary control knowledge for making the decisions we expect them to encounter. However acquiring this knowledge is difficult and time-consuming, and this knowledge alone does not always lead to robust performance in novel situations. Over the last year, we have done research on more general problem solving and planning approaches that can use more "fundamental" knowledge of the domain and thus increase the ability of Air-IFOR agents to respond to novel situations. Using experimental versions of TacAir-Soar, we have demonstrated the feasibility of integrating both look-ahead planning [van Lent, 1995] and means-ends analysis [Wray, 1995] into Air-IFOR agents.

In addition to the more general capabilities listed above, Air-IFOR agent must have knowledge that includes the doctrine and tactics appropriate to the missions they are to perform. Currently, Air-IFOR agents fly the following fixed-wing missions: BARCAP, Close Air Support, Strategic Attack, and MiGSweep. For rotary wing, Air-IFOR agents can fly a basic anti-armor mission [Tambe *et al.*, 1995b]. In addition, we have developed the following agents that act as controllers during missions [Nielsen, 1995].

- Air Intercept Controller (AIC) and Ground Controlled Intercept (CGI) which give information and commands about enemy planes. The AIC is situated in a plane with a large radar, such as an E2C.
- Forward Air Controller (FAC) which provides final directions for close-air support missions.
- Direct Air Support Center (DASC) assigns aircraft to missions, can change the mission, and hands off control to the FAC.
- Fire Support Coordination Center (FSCC) determines the type of support to utilize (close air support, artillery, or naval gunfire) and if close air support is determined it generates a tactical air request form then sends the request to the DASC.
- Tactical Air Command Center (TACC) which provides air traffic control, intermediate routing, and deconfliction.
- Tactical Air Direction (TAD) controller directs specific air operations within the area of operations, prior to the establishment of a DASC.

We have operational versions of all of these agents, although many are limited to producing behavior that is only relevant to close air support and air-to-air missions.

6 Multiagent Interactions

Although the individual agents are by themselves important, it is the coordination of agents that leads to effective military forces. Our approach is to model the methods and practices of military organizations. Air-IFOR agents coordinate their activities through a combination of common background knowledge (their knowledge of military methods, procedures, doctrine and tactics), common mission statements, and explicit communication (non-verbal and verbal) [Laird *et al.*, 1995]. Because Air-IFOR agents know what they are supposed to do and when (because of their background knowledge and mission statements), the need for explicit communication is greatly reduced. Also, in contrast to SAF agents, Air-IFOR agents are "smart" enough to deal with the details of executing all aspects of the missions they have been assigned and do not require constant monitoring by a human or command agent. When explicit verbal communication is used, we attempt to model both the content and form used by real pilots. Thus, Air-IFOR agents send simulated radio messages whose content closely mirrors the English words

and phrases used by real pilots. The generation and interpretation of these messages is currently done by a fixed set of templates and not a general-purpose natural language facility (although one is under development [Lehman *et al.*, 1995]). Air-IFOR agents currently can generate and interpret approximately 100 different types of messages.

When flying as a unit, most of the coordination occurs by the wingman visually observing and responding to the behavior of the lead of the unit. The wingman constantly adjusts its position to stay in the appropriate formation. The wingman also keeps track of the progress of the unit in its mission, observing the achievement of waypoints. Depending on the mission details, the wingman may change formation, break formation to fly an independent ground attack, rejoin the formation following an attack, or even take over as the lead.

Currently, TacAir-Soar agents (Air-IFOR agents for tactical fixed wing aircraft) are able to fly as either sections (two planes) or divisions (four planes). They can fly a variety of formations and they can dynamically break into smaller units, such as a division splitting into two sections, and then later reform as a single unit. Within a section, the lead and wingman can coordinate their radars (covering different parts of the sky and communicating enemy contacts) as well as coordinating their weapons employment during air-to-air engagments. During air-to-ground attacks, a section can use a variety of coordinated tactics, which are planned by the lead at the beginning of the mission. Our work on coordination with rotary wing units is also under development where currently the helicopters can fly in pairs, with the expected progression to platoons and then companies during the next year.

A unit of TacAir-Soar agents, such as a section or division, will also coordinate its behavior with available controllers (AIC, CGI, FAC, TAD, TACC, FSCC, DASC) [Nielsen, 1995]. The controllers can give the unit flight information (such as the altitude to fly at, or the name of the next controller), permission to continue the mission (permission to enter an area, or permission to attack a target), information on other planes, or changes to missions. In the case of changing a mission, a controller can dynamically change almost any aspect of a ground attack mission including the route, the time on target, and the final target. When a mission change is received, the members of the unit change their missions, sometimes replanning the final attack for air-to-ground missions.

Our goal is to continue to build up the coordination of Air-IFOR agents into integrated missions. We are currently close to completing close-air support which involves a variety of controllers plus planes doing individual missions. However, missions such as offensive strike and integrated interdiction can involve a variety of different planes flying many different individual missions (strategic attack, RECCE, MiGSweep, SEAD, etc.) that have to be closely orchestrated to pull off the complete mission. We plan on working on these missions and the required coordination over the next year.

Our approach to date has been to support the coordination of activities within the set of agents under our direct control. We have been able to develop our own templates independent of other groups. However, in the future some Air-IFOR agents will need to communicate with other command forces, and thus, we will soon be using CCSIL protocols for communication between our agents and their controllers.

7 STOW-E

During November 4–7, 1994, a large scale operational military exercise called STOW-E (Simulated Theater Of War – Europe) was held across 18 installations in United States and Europe. At its peak, over 1,800 entities were simulated on the Defense Simulation Internet (DSI). Although the vast majority of the entities were involved in ground actions, there were also a significant number of air missions being flown using humans in simulators, ModSAF agents, Soar/IFOR agents, and in a few cases, real planes with instrumentation that allowed them to be sensed within the DIS environment (although these planes could not sense the DIS entities). For the Soar/IFOR group, this was the first chance to participate in a realistic, large scale simulation environment where we did not have complete control over the scenarios.

Over the four day period, the Soar/IFOR agents were scheduled to participate in 10 events. For each event we had specific missions assigned to Air-IFOR agents that had been given to us weeks in advance. These missions included defensive air missions (BARCAPs), offensive air missions to disrupt BARCAPs, air to ground missions, and air to surface missions.

We successfully fielded agents for every event in which we were scheduled (10 events, approximately 32 agents) and participated in many unscheduled events (5–7 events, approximately 16 agents). TacAir-Soar performed air-to-air missions against ModSAF and humans (in simulators). TacAir-Soar attempted to engage planes from other sites, but because of problems with the network, the other agents did not see TacAir-Soar. We also participated in air-to-ground (bombing bridges, etc.) and air-to-surface (firing missiles at ships) attacks in which we engaged ground and surface targets from other sites.

We did have a limited number of software failures with the most significant being our inability to fly over the terrain database where the ground battle was raging when it was populated with hundreds of tanks. This was caused by a software bug in our C code for processing ground targets using radar.

One of our goals was to provide viable opponents for simulated and human pilots; however it was difficult to evaluate the "skill" of our TacAir-Soar because of some problems with the underlying simulation models. For example, during the first day, we were frustrated with the performance of TacAir-Soar in engagements. They were easily shot down by ModSAF F/A-18's. We later learned that in order to populate the simulation with different types of planes, the F/A-18's were created by copying F-14's. The F/A-18's were therefore carrying Phoenix missiles which are much longer range than any missile carried by an F/A-18. TacAir-Soar, basing its tactical behavior on the known properties of F/A-18's, was caught by surprise (as it should have been).

In engagements with humans, our planes would often get into good tactical positions, only to see our missiles miss when they were shot. (TacAir-Soar did have some kills against humans in simulators, but in general, TacAir-Soar got "toasted".) We believe that the missile missed because of flaws in the ModSAF missile models. Thus, although TacAir-Soar got shot down, it was in general using appropriate tactical maneuvers. Independent of the specific outcome, this exercise proved the value of taking systems out of the laboratory and testing them in more realistic situations.

Possibly the best example of our capabilities was in the execution of an unscheduled event for the second day. In this mission, a section of F/A-18's were to perform a ground attack against a set of islands in the simulated battle area. Our planes were used in place of a virtual (manned) ground attack because of the failure of that simulator. Enroute to the target, the planes were unexpectedly intercepted by ModSAF MiG-29's. The F/A-18's engaged the MiG-29's to defend themselves and got off one or two shots (but no kills). The MiG-29's disappeared from the network, and our planes automatically returned to their air-to-ground attack mission. Further enroute, they were unexpectedly fired on from a surface-to-air site, killing the wingman (not only did the planes not expect it, we didn't realize there would be any surface-to-air systems in STOW-E — clearly an unscripted interaction). The lead continued on, successfully dropping bombs on the designated target and then egressing back to base.

Although we considered our participation in this exercise a success, it did demonstrated some weaknesses that we must address in the future.

- Number of vehicles: We discovered that for an exercise with a large number of vehicles, we were not able to run the number of vehicles/workstation that we had expected. Part of this is the overhead in the network processing code of ModSAF, but it also was a problem for our AIC/E2C agent which could see a large number of agents at once because of its radar. This has led us to use more deliberate focusing of attention in Air-IFOR agents so that they do not attempt to process the complete situation at once, but instead concentrate on subsets of the situation, preferably those that are relevant to the current tactical situation.
- Mission set up: Before STOW-E, we had not developed any tools to help specify and manage the missions of Air-IFOR agents. During STOW-E, it was time-consuming and error-prone for us to create or modify the missions. As a result, we are currently developing graphical interface tools that will make it possible to enter and modify missions directly, without editing intermediate data structures. Our goal is that our interface should give the user the same look and feel as the documents and tools used by pilots in their normal briefings. The integration of Tcl and Soar is making this much easier because of its ability to manage windows and build formated graphical and textual interfaces. In the future we must also have the ability to accept missions from other software systems using CCSIL; however the details of the protocols have yet to be defined.
- Runtime control: Once Air-IFOR agents received their missions, they would fly the missions without any human management. Thus, we became observers and ran our exercises "hands-off". In contrast, the ModSAF planes required constant attention, with a human controlling their behavior on and off during the exercise. Although we wish to continue our approach, we also came to recognize that we needed the ability to dynamically change some aspects of the missions of Air-IFOR agents during the exercise, such as changing the waypoint at which a section of planes is stationed. These are relatively minor changes to TacAir-Soar.

This exercise has the additional significance of demonstrating that "hard core" AI technology can be successfully used in an operational exercise (although in STOW-E this was in a limited role). We believe that this is one of the first (if not the first) time that an AI system has been used in this way.

8 Summary and Conclusions

In the beginning of the Soar/IFOR project, there were many questions as to whether it was practical to develop intelligent forces for synthetic environments. Although there is still much more work to do, three years of research and development have brought us to the point where we can state with some degree of certainty that intelligent forces are practical and will play a significant role in STOW-97. It is difficult to isolate specific parts of our methodology or underlying technology as responsible for this success, although clearly we believe that the underlying Soar architecture is responsible to a significant degree. Its ability to combine fine-grain reactive reasoning of rules, with more deliberate and hierarchical decision making using operators within problem spaces, appears to be well matched to the demands of the interactive simulation and the cognitive processes of the humans we are attempting to model.

One surprise has been our ability to build complex and relatively general systems while not using many of the more advanced techniques such as means-ends analysis, planning, learning, complex agent modeling, or natural language. However, we still believe that these are critical capabilities for building robust, general agents, and we are continuing to pursue research in these areas.

In the immediate future, we will continue to expand the breadth of missions and capabilities of Air-IFOR agents. For fixed wing, a primary goal is to develop the appropriate agents to fly integrated interdiction and strategic attack missions. The coordination of many different types of aircraft, with different missions promises to be challenging. In rotary wing, our goal is to field a complete company of attack helicopters. Our plan is for these developments to lead up to a successful participation of Soar/IFOR agents in STOW-97.

9 Acknowledgments

This research was supported under contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory, and contract N66001-95-C-6013 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Command and Ocean Surveillance Center, RDT&E division. The authors would like to thank BMH Associates, Inc. for their technical assistance.

References

[Acharya and Tambe, 1993]
A. Acharya and M. Tambe. Collection-oriented match. In *Proceedings of the Second International Conference on Information and Knowledge Management*, November 1993.

[Calder *et al.*, 1993] R. Calder, J. Smith, A. Courtenmanche, J. Mar, and A. Ceranowicz. ModSAF behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[Doorenbos, 1994] R.B. Doorenbos. Combining left and right unlinking for matching a large number of learned rules. In *Proceedings of AAAI-94*, Seattle, WA, August 1994.

[Ernst and Newell, 1969] G. W. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving.* Academic Press, New York, 1969.

[Forgy, 1982] C. L. Forgy. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, 19:17–38, 1982.

[Hucka and Laird, 1995] M. Hucka and J. E. Laird. The Soar Development Environment. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

[Johnson, 1994] W.L. Johnson. Agents that learn to explain themselves. In *Proceedings of AAAI-94*, pages 1257–1263, Seattle, WA, August 1994. AAAI, AAAI Press.

[Jones *et al.*, 1993] R. M. Jones, M. Tambe, J. E. Laird, and P. S. Rosenbloom. Intelligent automated agents for flight training simulators. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 33–42, Orlando, FL, 1993.

[Kim and Rosenbloom, 1993] J. Kim and P. S. Rosenbloom. Constraining learning with search control. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 174–181, San Mateo, CA, 1993. Morgan Kaufmann.

[Koss and Lehman, 1994] F. V. Koss and J. F. Lehman. Knowledge acquisition and knowledge use in a distributed IFOR project. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 1994.

[Laird and Rosenbloom, 1994] J. E. Laird and P. S. Rosenbloom. The evolution of the Soar cognitive architecture. Technical report, Computer Science and Engineering, University of Michigan, 1994. To appear in Mind Matters, T. Mitchell Editor, 1995.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture

for general intelligence. *Artificial Intelligence*, 33(3), 1987.

[Laird *et al.*, 1995] J. E. Laird, R. M. Jones, and P. E. Nielsen. Multiagent coordination in distributed interactive battlefield simulations. Technical report, Computer Science and Engineering, University of Michigan, 1995.

[Lehman *et al.*, 1995] J. F. Lehman, J. Van Dyke, and R. Rubinoff. Natural language processing for IFORs: Comprehension and generation in the air combat domain. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Newell and Simon, 1956] A. Newell and H. A Simon. The logic theory machine: A complex information processing system. *IRE Transactions on Information Theory*, IT-2:61–79, September 1956.

[Nielsen, 1995] P. Nielsen. Intelligent computer generated forces for command and control. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Ousterhout, 1994] J. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA, 1994.

[Rosenbloom *et al.*, 1991] P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47, 1991.

[Rosenbloom *et al.*, 1993] P. S. Rosenbloom, J. E. Laird, and A. Newell. *The Soar Papers: Research on Integrated Intelligence.* MIT Press, 1993.

[Rosenbloom *et al.*, 1994] P. S. Rosenbloom, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, J. F. Lehman, R. Rubinoff, K. B. Schamb, and M. Tambe. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, May 1994.

[Salisbury, 1995] M. Salisbury. Command and Control Simulation Interface Language (CCSIL): Status update. In *Proceedings of the 12th Distributed Interactive Simulation Workshop*, 1995. Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

[Schwamb *et al.*, 1994] K. B. Schwamb, F. V. Koss, and D. Keirsey. Working with ModSAF: Interfaces for programs and users. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, May 1994.

[Tambe and Rosenbloom, 1995] M. Tambe and P. S. Rosenbloom. Agent tracking in complex multi-agent environments: New results. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[Tambe *et al.*, 1995a] M. Tambe, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 1995.

[Tambe *et al.*, 1995b] M. Tambe, K. Schwamb, and P. S. Rosenbloom. Building intelligent pilots for simulated rotary wing aircraft. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 1995.

[van Lent, 1995] M. van Lent. Planning and learning in a complex domain. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

[Wray, 1995] R. E. Wray. A general framework for means-ends analysis. Technical report, The University of Michigan, Department of Electrical Engineering and Computer Science, 1995.

10 Biographies

John E. Laird is an associate professor of Electrical Engineering and Computer Science and the director of the Artificial Intelligence Laboratory at the University of Michigan. He received his B.S. degree in Computer and Communication Sciences from the University of Michigan in 1975 and his M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University in 1978 and 1983, respectively. His interests are centered on creating integrated intelligent agents (using the Soar architecture), leading to research in problem solving, complex behavior representation, machine learning, cognitive modeling.

W. Lewis Johnson is a project leader at the University of Southern California Information Sciences Institute, and a research assistant professor in the USC Department of Computer Science. Dr. Johnson received his A.B. degree in Linguistics in 1978 from Princeton University, and his M.Phil. and Ph.D. degrees in Computer Science from Yale University in 1980 and 1985, respectively. He is interested in applying artificial intelligence techniques in the areas of computer-based training and software engineering. Dr. Johnson is co-editor-in-chief of the journal Automated Software Engineering, Secretary/Treasurer of the

SIGART Bulletin, and is on the steering committee of the AI and Education Society.

Frank V. Koss is a systems research programmer in the Artificial Intelligence Laboratory at the University of Michigan, where he is developing the interface between the Soar cognitive architecture and the ModSAF simulator and extending ModSAF itself. He received his BS in computer engineering from Carnegie Mellon University in 1991 and his MSE in computer science and engineering from the University of Michigan in 1993.

Jill Fain Lehman is a research computer scientist in Carnegie Mellon's School of Computer Science. She received her B.S. from Yale in 1981, and her M.S. and Ph.D. from Carnegie Mellon in 1987 and 1989, respectively. Her research interests span the area of natural language processing: comprehension and generation, models of linguistic performance, and machine learning techniques for language acquisition. Her main project is NL-Soar, the natural language effort within the Soar project.

Randolph M. Jones received his Ph.D. in Information and Computer Science from the University of California, Irvine, in 1989. He is currently an assistant research scientist in the Artificial Intelligence Laboratory at the University of Michigan. His research interests lie in the areas of intelligent agents, problem solving, machine learning, and psychological modeling.

Paul E. Nielsen is an assistant research scientist at the Artificial Intelligence Laboratory of the University of Michigan. He received his Ph.D. from the University of Illinois in 1988. Prior to joining the University of Michigan he worked at the GE Corporate Research and Development Center. His research interests include intelligent agent modeling, qualitative physics, machine learning, and time constrained reasoning.

Paul S. Rosenbloom is an associate professor of Computer Science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated intelligent systems (in particular, Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councilor and Fellow of the AAAI and a past Chair of ACM SIGART.

Robert Rubinoff is a postdoctoral research fellow in Carnegie Mellon's School of Computer Science. He received his B.A., M.S.E., and Ph.D. from the University of Pennsylvania in 1982, 1986, and 1992, respectively; his dissertation research was on "Negotiation, Feedback, and Perspective within Natural Language Generation". His research interests include natural language processing, knowledge representation, and reasoning. He is currently working on natural language generation within the Soar project.

Karl B. Schwamb is a Programmer Analyst on the Soar Intelligent FORces project at the University of Southern California's Information Sciences Institute. He contributes to the maintenance of the Soar/ModSAF interface software and the Tcl/Tk interface to Soar. He received his M.S. in Computer Science from George Washington University.

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC) and a research assistant professor with the Computer Science Department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, India in 1986. He received his Ph.D. in computer science from Carnegie Mellon University in 1991. His interests are in the areas of integrated AI systems, agent modeling, plan recognition, and efficiency and scalability of AI programs, especially rule-based systems.

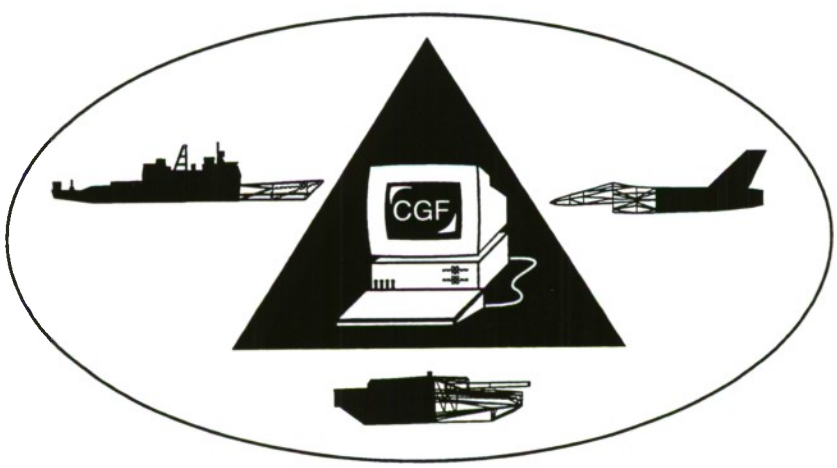
Julie Van Dyke is a research programmer at Carnegie Mellon University, working on language comprehension in NL-Soar. She is also working toward an MS in Computational Linguistics with a focus on modeling language acquisition.

Michael van Lent is currently a doctoral candidate in the Artificial Intelligence Laboratory at the University of Michigan. He received his B.A. with honors in computer science from Williams College in 1991 and a Master of Science in Computer Science from the University of Tennessee, Knoxville in 1993. Mr. van Lent also worked for the Naval Center for Applied Research in Artificial Intelligence during the summers of 1992 and 1993.

Robert Wray is currently a candidate for the Ph.D degree in computer science at the University of Michigan. He received a Bachelor of Science in Electrical Engineering from Memphis State University in 1988 and a Master of Science in Electrical Engineering from the University of Massachusetts, Dartmouth in 1993. He also worked for the Naval Undersea Warfare Center from 1989 to 1993 as an electronics engineer. His current research, projects and interests in computer science include: intelligent agent architectures, extending traditional artificial intelligence planning paradigms, machine learning, and software engineering.

Session 2b: Reasoning I

Tambe, ISI, USC
Ge, Cornell University
Gonzalez, Dept. of Elec.& Comp. Eng., UCF
Kocabas, Marmara Research Center


CGF

Building Intelligent Pilots for Simulated Rotary Wing Aircraft

Milind Tambe, Karl Schwamb and Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
email: {tambe, schwamb, rosenbloom}@isi.edu

1. Abstract

The Soar/IFOR project has been developing intelligent pilot agents (henceforth IPs) for participation in simulated battlefield environments. While previously the project was mainly focused on IPs for fixed-wing aircraft (FWA), more recently, the project has also started developing IPs for rotary-wing aircraft (RWA). This paper presents a preliminary report on the development of IPs for RWA. It focuses on two important issues that arise in this development. The first is a requirement for reasoning about the terrain — when compared to an FWA IP, an RWA IP needs to fly much closer to the terrain and in general take advantage of the terrain for cover and concealment. The second issue relates to code and concept sharing between the FWA and RWA IPs. While sharing promises to cut down the development time for RWA IPs by taking advantage of our previous work for the FWA, it is not straightforward. The paper discusses the two issues in some detail and presents our initial resolutions of these issues.

2. Introduction

The Soar/IFOR project has been developing intelligent pilot agents (IPs) for simulated battlefield environments (Laird et al., 1995, Rosenbloom, et al., 1994, Tambe et al., 1995). Until Summer 1994, the project was focused on building IPs for simulated fixed-wing aircraft (FWA), including air-to-air fighters and ground-attack aircraft. Since July 1994, we have begun developing IPs for simulated rotary-wing aircraft (RWA), specifically, AH-64 Apache attack helicopters.

While there are similarities in an RWA and an FWA pilot's missions — e.g., employing weapons on targets, flying mission-specified routes — there are also some important differences. One key difference is reasoning about the terrain. For example, an RWA pilot's mission can involve flying Nap-of-the-earth (NOE), where it needs to fly only about 25 feet above ground level, while avoiding obstacles. It may also involve flying through a valley, or around a forested region. The mission may also involve hiding (masking) behind a ridge, popping up to spot enemy targets, and remasking in a new hiding position. Figure 1 provides an illustration of this type of terrain reasoning. It presents a snapshot, taken from ModSAF's plan-view display (Calder et al., 1993), of a typical scenario involving Soar-based RWA IPs. There are two RWA in the scenario, just behind the ridge, indicated by the contour lines. The other vehicles in the figure are a convoy of "enemy" ground vehicles — tanks and anti-aircraft vehicles — controlled by ModSAF. The RWA are approximately 2.5 miles from the convoy. The IPs have hidden their helicopters behind the ridge (their approximate hiding area is specified to them in advance). They unmask these helicopters by popping out from behind the ridge to launch missiles at the enemy vehicles, and quickly remask (hide) by dipping behind the ridge to survive retaliatory attacks. They subsequently change their hiding position to avoid predictability when they pop out later.

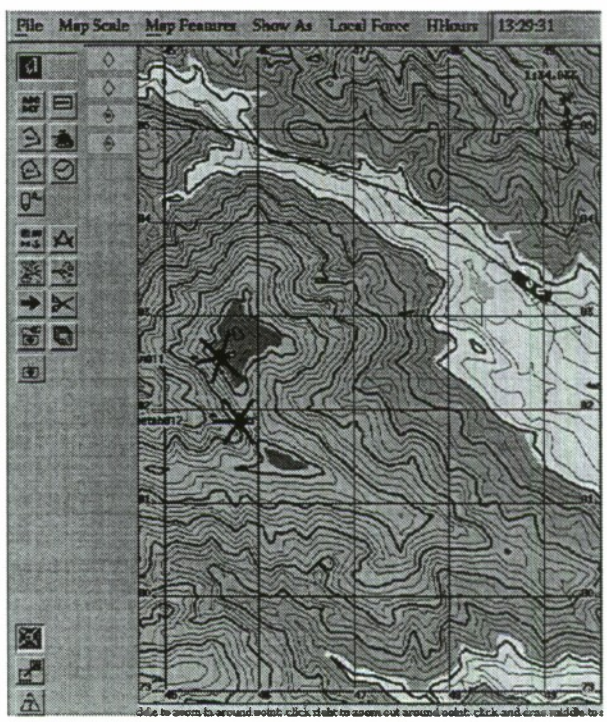


Figure 1: A snapshot of ModSAF's simulation of an air-to-ground combat situation.

Thus, the development of RWA IPs brings up the novel issue of terrain reasoning, not addressed in previous work on Soar/IFOR agents. There has been much work on terrain reasoning in ModSAF in their development of semi-automated forces or SAFs

(Calder et al., 1993). That work has so far primarily focused on ground-based SAFs (e.g., (Longtin, 1994)), although there is a recent effort focused on terrain reasoning for RWA (Tan, 1995). Outside the arena of automated forces, terrain reasoning in the form of route planning and execution has been addressed extensively in AI and Robotics. The focus of much of this work is on 2D routes (Denton and Froeberg, 1984, Khatib, 1986, Lozano-Perez and Wesley, 1979, Mitchell, 1990) — and this category includes some previous work within Soar (Stobie et al., 1992) — although some efforts have also attacked the 3D route planning problem (Bose et al., 1987, Rao and Arkin, 1989). Other aspects of terrain reasoning such as tactical situation assessment (McDermott and Gelsey, 1987) and hiding (Stobie et al., 1992) have also received some attention, although not nearly as much as route planning. As discussed in Section 3, the pure route planning approaches from this literature are unlikely to address the terrain reasoning challenge facing the RWA IPs, which is to accomplish these tasks in real-time, given a realistic 3D terrain database. A hybrid solution combining some abstract plans with reactivity is currently being investigated.

Given the similarities between the FWA and RWA IPs, concept and code sharing between the two is a real possibility. Sharing would speed up development of RWA IPs by taking advantage of our previous work on FWA. However, the differences — such as the terrain reasoning capability above — imply that sharing is not straightforward. There have been some previous efforts aimed at facilitating reuse of code and concepts among Soar systems. These efforts have typically focused on reuse of individual capabilities, such as inductive learning (Rosenbloom and Aasman, 1990), or natural language (Lewis, 1993, Rubinoff and Lehman, 1994) capabilities. The novel issue here is that a large fraction of the FWA IP structure is potentially reusable in developing RWA IPs and such reuse needs to be facilitated.

The rest of this paper provides more details on these two issues. Section 3 focuses on terrain reasoning. Section 4 discusses the issue of code and concept sharing between Soar-based FWA and RWA IPs. We will assume some familiarity with the Soar architecture (Laird, Newell, and Rosenbloom, 1987, Rosenbloom, et al., 1991).

3. Terrain Reasoning

The overall terrain reasoning tasks for an RWA IP may be subdivided into two categories. The first is to fly from a given source to a destination, while abiding by mission specified constraints regarding the flight methods. A flight method primarily specifies maintenance of a certain air-speed and altitude above ground level. In particular, a *high-level* flight requires that the RWA fly more than 200 feet above ground level with air-speed as high as 145 knots. A *low-level* flight requires that the RWA fly 100-200 feet above ground level, while maintaining a maximum air-speed of 100 knots. A *contour* flight requires the RWA to fly between 25-100 feet above ground level, but with a maximum air-speed of 70 knots. An *NOE* flight requires the RWA to fly within just 25 feet above ground level, with a maximum air-speed of 40 knots. Additionally, an NOE flight may require that an RWA fly through a valley along a hillside, or through a narrow clear corridor in a forested region. The second category of terrain reasoning tasks involves an RWA IP's activities once it successfully follows its route to its battle area, and possibly engages enemy vehicles. Its activities in this area involve selecting and occupying good hiding positions (behind a ridge or a forested region) and flying between hiding positions while remaining concealed from a possibly mobile enemy. It may also involve reasoning about possible enemy hiding positions.

For both categories of tasks, one key issue for an RWA IP is to execute them in the context of a large-scale and realistic 3D terrain database, with features such as rivers, ridges, valleys, hills and forested regions. A second key issue is that given its complexity, the cost of sensing and processing large tracts of the terrain database is non-trivial. A third related issue is that an IP has to exhibit human-like behavior in performing these terrain reasoning tasks. Thus, it should not make use of information that a human pilot is unlikely to obtain. For example, as with a human pilot, an IP should plan routes using a map of the terrain database (which possibly may be inaccurate), rather than using the actual terrain database (which would always be 100% accurate). A final issue is that an IP has to perform its tasks in real-time. The following two subsections illustrate how these issues are addressed for each of the two types of tasks above.

3.1. Route Flying

For the task of route flying, one possible approach for addressing the above issues would be to use one of a variety of path-planning methods that provides a very detailed 3D point-to-point route, with little need or freedom to modify the given route (Stobie et al., 1992, Bose et al., 1987, Rao and Arkin, 1989, Denton and Froeberg, 1984). One such approach, based on weighted-region path planning (Mitchell, 1990), is to conceptually divide a map of the terrain into 3D cells (cubes), assign an appropriate cost to each cell that reflects mission-specified constraints, and then search for a minimum cost path through the cells. One advantage of such an approach is that an RWA IP need not sense the terrain database in any detail, but rather just enough to follow its plan. In addition, the

low sensing overhead would facilitate real-time task performance. However, there are several problems with such an approach. First, given the complexity of the terrain, this approach would require a significant initial computational effort to create and then search the cells. Second, it could be wasteful given the realism of the RWA model and its flight controls — it will not be possible for a Soar-based IP to precisely control an RWA to follow such a detailed route, and it will end up having to reactively improvise the path or replan. The original planner could potentially take these realistic flight controls into account when developing a plan — so that no on-line replanning may be required — but that would further increase the complexity of planning. Third, if the map of the terrain is inaccurate or incomplete, the plan generated could be inaccurate as well. Even if the map were completely accurate (or if the IP were using the terrain database itself rather than a map), there could still be deviations from the planned route caused by an unexpected encounter with hostile or friendly vehicles. Thus, an IP may not be able to rely on just its original planned route; it may need to replan. Finally, human pilots typically do not rely on such detailed plans; and thus in forcing IPs to follow such plans, we are likely deviating from our goal of building human-like IPs.

So instead, a Soar-based IP follows a hybrid strategy that combines a plan-based and reactive strategy. In particular, it relies on more abstract route plans, that provide it just two to three intermediate points.[1] The IP then executes these route plans while reacting to sensory information that enables it to abide by the mission specified constraints. For ideal human-like IPs, this sensory information should be precisely what a human pilot would obtain visually by looking out the window. Unfortunately, for an IP, such visual processing is likely to be extremely complex and expensive. Therefore, special inexpensive sensors have been designed that approximate such visual processing. One such sensor is the *look-ahead altitude sensor* or *LAS* sensor. LAS is slaved to the parameters supplied by the IP. The IP sets parameters for LAS that specify a lookahead range and orientation, which in turn specifies a line segment of specific length and orientation originating from the IP's current location. Once these parameters are set, LAS scans the terrain database repeatedly (in fact, each time ModSAF schedules the agent for execution), and returns the highest altitude value along the specified line segment. For instance, to fly NOE, an IP sets LAS's parameters to a lookahead range of 50 meters, and orientation in the direction of its flight. The pilot reacts to LAS's response by modifying the altitude of its helicopter to be approximately 25 feet above the highest point.[2]

The top half of Figure 2 shows a pilot agent making use of LAS to fly NOE. The shaded portion in the figure is a profile of the terrain, while the dashed line is a profile of the helicopter flying NOE. The straight lines indicate LAS's lookahead range while scanning the database. The bottom half of Figure 2 indicates a longer lookahead range, and change in the flight profile that that results.

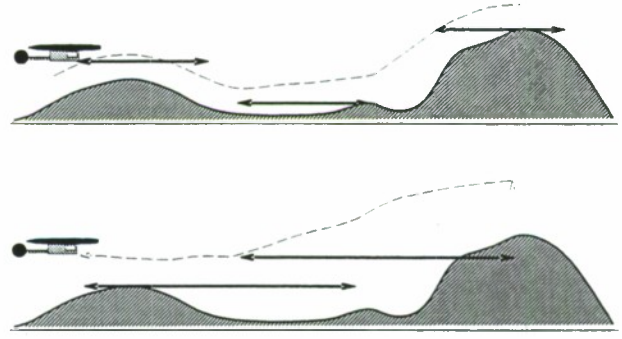


Figure 2: Illustrations of lookahead altitude sensor. LAS scans the terrain database each time the agent is scheduled for execution (illustrations not from an actual run).

The precise value of the lookahead range is determined to a large extent by the speed of the RWA. In particular, for an NOE flight, an IP currently flies conservatively at a speed of 20 knots. With 50 meters lookahead, that gives it about 5 seconds to change its altitude. The other flight methods, specifically contour, low-level and high-level flight, require that the RWA fly at a higher speed. This in turn requires that the IP set a longer lookahead range to give itself more time to react. Speed is however not the only factor determining the lookahead range. It is also dependent on the type of flight profile desired. For instance, at its speed of 80 knots, an IP could potentially sustain the altitude required for its low-level flight with a lookahead of just 200-300 meters. However, the flight profile generated follows the terrain much too closely — it is not as smooth as the flight profile that results from a human pilot's low-level flight (at least as indicated by the experts). Therefore, the low-level flight uses a much longer lookahead range of 1500 meters. The high-level flight uses a lookahead range of 5000 meters.

Unfortunately, long lookahead ranges in LAS could potentially hinder an IP's real-time performance. Therefore, to lower its cost, LAS samples precisely 100 points along the specified line

[1] At present, these abstract routes are provided by a human; although given that they are abstract, planning these routes is expected to be much less complex.

[2] RWA agents in ModSAF appear to follow a similar technique (Tan, 1995).

segment irrespective of the lookahead range. Thus, despite the variation in the lookahead range in Figure 2, LAS will scan precisely 100 points. This sampling resolution may appear to be very low, with the potential of missing high altitude cliffs. However, LAS's repeated scanning in effect improves its sampling resolution. In particular, since an RWA progresses towards its destination between two scans, successive scans sample slightly different points. In fact, each successive scan samples 99 points in the neighborhood of the points from its previous scan (on the same line segment), and one new point. This resolution could still be insufficient for some types terrain. For instance, if the terrain is an urban landscape with a sparse population of pin-shaped high-altitude structures,[3] there is a small possibility that LAS may miss those in its scanning. In such cases, there may be a need to increase the sampling resolution. However, the 100 point scans have so far proved adequate over the terrain database used in our experiments (the RWA have not crashed).

Figure 3 presents a flight profile from an actual run of a Soar-based RWA using the contour flight method. Figure 4 presents a flight profile from another run of a Soar-based RWA over approximately the same terrain, but using the NOE flight method. The shaded portion indicates the terrain, while the dashed line indicates the actual flight profile. IPs smoothen out the flight by using fuzz-boxes (McDermott and Davis, 1984) to avoid excessive altitude adjustments.

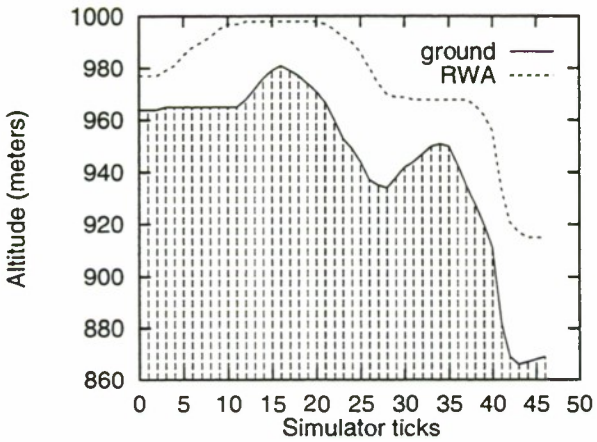


Figure 3: Illustration of a contour flight from an actual run.

Similar low-cost, LAS-type sensors approximating a human pilot's visual input are currently being designed to enable the RWA pilots to fly through valleys.

[3]A clock tower would be one example of such a structure.

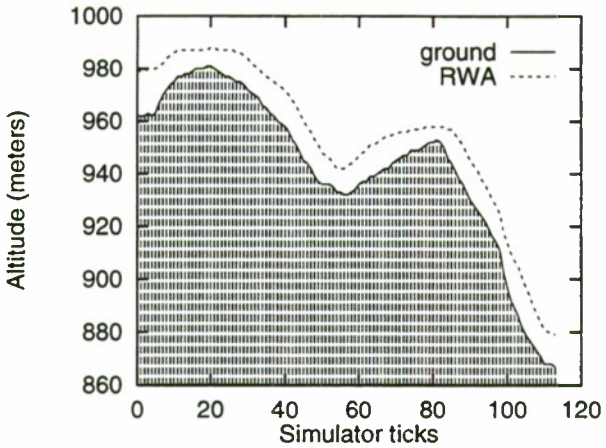


Figure 4: Illustration of an NOE flight from an actual run.

3.2. Hiding

Once an RWA IP reaches its mission-specified battle area, it needs to engage in hiding-related tasks. In general, a battle area could be of an arbitrary (convex) shape, or specified in terms of landmarks, such as trees or rocks. The IP should be capable of locating good hiding positions within this area and move between hiding positions while remaining concealed from its enemy. This second terrain reasoning capability, at least at this level of generality, is very much an issue for future research. At present, we have restricted the battle area to be a rectangle. One side of this rectangular area, typically coinciding with a ridge or a tree line, is a mission specified line segment. This is in essence considered to be an imaginary wall, and any movement behind it is assumed to be hidden from the enemy. An RWA IP hides in a small rectangular area (defined with a width of 100 meters) behind this imaginary wall. When relocating to a new hiding position, it uses the NOE flight method to remain at a low altitude and thus hidden behind the wall. The approximations of a wall and a rectangular area for hiding are both based on our previous work in the *groundworld* domain. Groundworld involved a simulated terrain with random configurations of horizontal and vertical walls, where an intelligent agent had to hide behind a wall to escape from another agent pursuing it (Stobie et al., 1992, Tambe and Rosenbloom, 1993).

4. Sharing and Reuse

RWA pilots' missions have some requirements — such as, identifying enemy vehicles, firing missiles at target vehicles and flying in formation — in common with those of FWA pilots. These commonalities may be exploited to cut down development time by sharing or reusing both code and concepts from Soar-based FWA pilots in the development of RWA pilots. For instance, for an FWA IP, the code for firing a

missile involves three operators that orient its aircraft towards its target, then push a fire button to actually launch the missile, and then guide the missile (should the missile require guidance) via radar (or other) illumination of the target. These three operators can be reused in an RWA IP. At present, a Soar-based RWA IP has 44 operators, with 25 (that is about 57%) reused in some form from the Soar-based FWA IPs. The 19 new operators are those involved with terrain reasoning tasks such as flying NOE, masking and unmasking. This sharing is accomplished simply by loading in appropriate operators from an FWA IP code in an RWA IP.

Differences in concepts and terminology, however, make some of the sharing problematic. For example, for FWA pilots engaged in air-to-air missions, the concept of launch-acceptability-region or LAR of a missile combines both the range to a target and the target aspect (angle between the target's current heading and the straight line joining the target and the FWA pilot's current locations). Thus, if a target is heading towards the FWA pilot with a 0° target aspect, the missile may be fired from a long range; but the range is reduced substantially if the target has a 180° target aspect. In contrast, for an RWA pilot, the target aspect is irrelevant in calculating a missile's LAR — the missile may be fired at an equally long range irrespective of the target aspect. This creates a significant difference in the concept of a missile LAR for an FWA and an RWA IP, making the sharing of missile-LAR-related code difficult. There is an accompanying difference in the terminology as well — the RWA pilot refers to the missile LAR as a missile constraint.

At least some of these apparent discrepancies in the two IP's concepts — and potentially their terminology — could be resolved if the agents reason about the concepts from first principles. For instance, agents could calculate a missile's LAR from first principles, based on the relative velocities (speed and direction) of the missile and the target. Since an FWA IP's target in air-to-air combat is a fighter jet, moving at a speed that may be only a half to a fifth its missile speed, its angle of movement (target aspect) becomes an important factor in calculating LAR. In particular, a target moving towards the FWA allows a missile to be fired from a much longer range; while a target that is moving away requires that the missile be fired from a much closer range, so that the missile may catch up with the target before expending all its fuel. In contrast, an RWA IP's target is moving two orders of magnitude slower than its missile — the angle of the target's movement has a negligible impact on the missile range. In other words, with the first principles calculations, the target aspect discrepancy automatically disappears. It will appear important in FWA IP's calculations, and negligible in an RWA IP's calculations.

While such calculations from first principles would facilitate sharing, the calculations themselves may be prohibitively expensive, and hinder real-time performance. Soar's chunking (learning), could potentially compile such first principles calculations into new rules and alleviate this cost. However, that remains an issue for future work. We are currently relying on a lower cost alternative, where a problematic aspect of the agent code is rewritten when in reuse.

5. Current Status and Future Work

As of February 1995, the RWA agents are capable of performing a complete *attrit* mission, which involves flying to a battle area using one of the possible flight methods, followed by masking, unmasking, firing missiles at targets, and relocating to a different masking location between missile firings. We have run scenarios with up to four RWA IPs executing the *attrit* mission.

At present the RWA IPs can fly in coordination, in pairs. Extending this work to enable coordinated mission execution involving a platoon or a company of RWA agents (with a platoon and a company commander), is at the top of our agenda for future work. Agents at higher echelons of command, such as a company commander, may also bring up issues of communication and mission planning, which we have currently not addressed. Other issues for future work, mentioned in previous sections, include improvement in terrain reasoning for hiding, and in code/concept sharing among Soar agents.

6. Acknowledgements

This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL); and under contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD). Critical expertise and support has been provided by David Sullivan of BMH Inc.

7. References

Bose, P. K., Meng, A. C-C., Rajnikanth, M. (1987) Planning flight paths in dynamic situations with incomplete knowledge. Proceedings of the SPIE conference on Spatial reasoning and multi-sensor fusion.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z. (1993) ModSAF behavior simulation and control. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Denton, R. V., and Froeberg, P. L. (1984) Applications of Artificial Intelligence in Automated Route Planning. Proceedings of SPIE conference on applications of Artificial Intelligence. , pp. 126-132.

Khatib, O. (1986) "Real-time obstacle avoidance for manipulators and mobile robots". *International Journal of Robotics Research 5*, 1 , 90-98.

Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K., Tambe, M., van Lent, M., and Wray, R., (May, 1995) Simulated Intelligent Forces for Air: The Soar/IFOR project 1995. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.

Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987) "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 , 1-64.

Lewis, R. L. (1993) An architecturally-based theory of human sentence comprehension. Proceedings of the Eleventh Annual Conference of the Cognitive Science Society.

Longtin, M. J. (1994) Cover and concealment in ModSAF. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Lozano-Perez, T. and Wesley M. A. (1979) "An algorithm for planning collision-free paths among polyhedral obstacles". *Communications of the ACM 22*, 10 , 560-570.

McDermott, D. and Davis, E. (1984) "Planning routes through uncertain territory". *Artificial Intelligence 22* , 107-156.

McDermott, D., and Gelsey, A. (1987) Terrain analysis for tactical situation assessment. Proceedings of the SPIE conference on Spatial reasoning and multi-sensor fusion.

Mitchell, J. S. B. (1990) Algorithmic approaches to optimal route planning. Proceedings of the SPIE conference on Mobile Robots.

Rao, T. M., and Arkin, R. C. (1989) 3D Path planning for flying/crawing robots. Proceedings of the SPIE conference on Mobile Robots.

Rosenbloom, P.S. and Aasman J. (August, 1990) Knowledge level and inductive uses of chunking (EBL). Proceedings of the National Conference on Artificial Intelligence. , pp. 821-827.

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. (1991) "A preliminary analysis of the Soar architecture as a basis for general intelligence". *Artificial Intelligence 47*, 1-3 , 289-325.

Rosenbloom, P., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Lehman, J. F., Rubinoff, R., Schwamb, K., and Tambe, M. (1994) Intelligent Automated Agents for Tactical Air Simulation: A Progress Report. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Rubinoff, R., and Lehman, J. (1994) Natural language processing in an IFOR pilot. Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.

Stobie, I., Tambe, M., and Rosenbloom, P. (Novemeber, 1992) Flexible integration of path-planning capabilities. Proceedings of the SPIE conference on Mobile Robots.

Tambe, M., and Rosenbloom, P. (July, 1993) On the Masking Effect. Proceedings of the National Conference on Artificial Intelligence.

Tambe, M., Johnson, W. L., Jones, R., Koss, F., Laird, J. E., Rosenbloom, P. S., and Schwamb, K. (Spring 1995) "Intelligent agents for interactive simulation environments". *AI Magazine 16* .

Tan, J. Flying NOE in ModSAF. Private communication.

8. Authors' Biographies

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC) and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, India in 1986. He received his Ph.D. in computer science from Carnegie Mellon University in 1991. His interests are in the areas of integrated AI systems, agent modeling, plan recognition, and efficiency and scalability of AI programs, especially rule-based systems.

Karl Schwamb is a Programmer Analyst on the Soar Intelligent FORces project at the University of Southern California's Information Sciences Institute. He contributes to the maintenance of the Soar/ModSAF interface software and the Tcl/Tk interface to Soar. He received his M.S. in Computer Science from George Washington University.

Paul S. Rosenbloom is an associate professor of computer science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated intelligent systems (in particular, Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councillor and Fellow of the AAAI and a past Chair of ACM SIGART.

A Multiple Agent Hybrid Control Architecture for Automated Forces: Design and Software Implementation

Xiaolin Ge, John James, and Anil Nerode
Mathematical Sciences Institute of Cornell University and Sagent Corp.
xge@math.cornell.edu, 75767.77@compuserve.com, anil@math.cornell.edu

1. Abstract

We introduce a multiple-agent hierarchical command system for automated forces based on distributed optimization of local cost functions. Issues of system design and software implementations are discussed.

2. Introduction

We introduce a MAHCA, or mutiple agent hybrid control architecture, for generation of computer-generated forces (CGF) and for use with semi-automated forces (SAF) \cite{mm94}. This architecture has already been applied to many other areas [13] [13] [17] [14], but each new application area presents new intellectual modelling challenges. Here we provide (1) an overview of ideas for apply existing MAHCA technology to overcoming technical barriers in CGF and SAF [24] and (2) a summary of results from a previous demonstration of use of MAHCA for generation of software which synchronizes logical behavior and continuum behavior of units involved in engaging multiple targets with multiple weapons platforms.

As discussed in [23] and [22], activities in the area of doctrine-based software design are meant to ensure that the final SAF performance behaviors accurately reflect current Army doctrine. Achieving this goal is made more difficult by the fact that, even though doctrinal publications exist and are constantly updated, doctrine development is itself an ongoing, dynamic process. Additionally, even though the Army Materiel Systems Analysis Activity (AMSAA) has accurate models for many low-level, continuum activities, the integration of doctrinal constraints on CGF behaviors with continuum constraints on CGF behaviors remains an experimentation-based design process. The current approach for capture of battlefield activities in support of combat maneuvers is to use Army Training and Evaluation Program (ARTEP) documents to support creation of Combat Instruction Sets (CISs). A Combat Instruction Set (CIS) is defined as a computer-generated representation of tactical combat behavior at unit (Battalion, Company, Platoon, Squad, Section, and Fire Team) and platform (ground vehicle, air vehicle, or dismounted infantry/engineer/scout entities at various levels of organization) level [22].

At a recent meeting on research challenges in DIS the increasingly difficult problem of achieving interoperability between heterogeneous visual systems such as Battlefield Distributed Simulation - Developmental (BDS-D) and Close Combat Tactical Trainer (CCTT) was discussed by a representative of the U. S. Army Simulation, Training and Instrumentation Command (STRICOM) [see [19] 9, p. 270]. At that same meeting Kohn, Nerode, and James discussed the application of MAHCA technology to advanced distributed simulation [21]. Here we augment that discussion with details concerning an initial experiment in building a multiple-agent simulation. While not intended to capture the full complexity of the engagement process, the experiment did show the feasibility of using a declarative, formal approach for synchronization of logical and continuum models required for construction of a CIS. Also, since the MAHCA technology applies a formal extraction methodology, the experiments needed for the Verification, Validation and Accreditation (VV&A) process for commissioning a CIS can be reduced. Finally, since MAHCA technology is declarative, the process of building the composite logical and continuum models is incremental, which supports rapid inclusion of doctrinal changes in CGF applications.

3. MAHCA Agents for Automated Forces

The MAHCA Architecture assigns a software agent to each unit (Fire Team, Squad, Platoon, ...) of a complex system which consists of a large number of interacting units. Each agent observes its unit, changes agent state as a function of the state of its unit and of messages passed from otheroptimization algorithm to extract a plan which comes close to minimizing the agent local cost function. Every agent's cost function is periodically updated by adding an adjustment term communicated from every other agent. This is one of

two forms of communication between the agents. The other is communication of constraints issued by agents higher in the hierarchy of agents. The agents otherwise compute autonomously. The plans the agent issues are advisory for its unit and, if followed, constrains the actions of that unit until its agent issues a new plan. We use a simple form of optimization here. More complexoptimizations, based on chattering and the measure valued calculus of variation are the basis of general MAHCA, but are not employed in this paper.

The agents have varying degrees of autonomy, but all have to obey the constraints of the system as expressed by doctrine, physical circumstances and laws, and also the constraints imposed by higher commands. We model a military command as a hierarchy of agents. Higher agents in the hierarchy can issue plans constaining the lower agent's behaviors. Starting bottom-up, each asset such as an airplane, tank, or scout vehicle is viewed as under the supervision of a corresponding agent. Similarly with the human element. A platoon has an agent for each Squad and a higher level agent, for the Platoon Leader, each with its own cost function.

The system can be modelled by building agents bottom-up from the lowest military unit up to the highest level of command. One new feature is that every agent has its own mathematical model of the local unit and its own local cost function, and is performing an optimization based on that model and cost function and terms from other agents and constraints from higher agents. A second new feature is that near-optimal plans are extracted by each agent and given to its unit. A simulation based on this methodology is not limited to answering ``what if'' statements after imposing strategies that arise from the outside, but can plan close to optimal strategies itself. A third advantage is that since a unit's cost function is what counts for determining unit interactions, aggregated units are fully compatible in the model with less aggregated units, provided only that the cost functions are properly adjusted.

This is a fully scalable architecure for complex heterogeneous systems. Since we delegate as much computation as possible to local units to minimize interagent communications, this architecture is quite suitable for distributed simulation. It can be implemented by putting one computer for MACHA at each simulator which can monitor and replace inputs and outputs of its simulator, and can communicate directly with other agents, plus with agents for the command hierarchy. It is fully compatible with incorporating legacy simulators of all kinds, constructive, virtual, etc., because its fully integrated communications system between agents is there to take care of all impedance and phase mismatches between simulators to the extent this can be done at all.

3.1 Agents

What did our platoon level agent program incorporate?

(1) Each tank squad has an agent of the same lowest hierarchy level. These are fellow agents. There is a platoon leader agent at the next higher level in the hierarchy. While it is feasible to create an agent for each squad member, we did not do so for our simulation experiment.
(2) The agent has a sensor system.
(3) The agent contains a mathematical model of the system.

There are three sets of variables on the basis of which plans are made.

(a) Variables correponding to readings of the agents sensor systems. These will be updated through online sensor readings. This group is updated as often as feasible.
(b) Variables corresponding to information passed from other agents. These are not raw data, but assessed information which represent the other agent's judgements on the state of the system.
(c) Variables corresponding to orders passed down by the platoon leader agent.

There is no control exercised between fellow agents. Cooperation is achieved through information passing between agents which is about the global state of the platoon. The Platoon Leader agent can enforce policy changes by adjusting the third set of parameters in the mathematical formula that is used to compute plans.

(4) The platoon model has an information system and database.

The platoon leader agent has the same design. The plan of the platoon leader is given to the platoon agents. Similarly one can build extended hierarchies of agents corresponding to higher levels of command.

In the sections that follow we give a simple example of a multiple agent reactive decision system which we implemented at Picitinny Arsenal with the cooperation of Dr. Norman Coleman.

This was to demonstrate MAHCA multiple agent distributed control for a platoon of simulated tanks.

4. Describing a Decision Problem

The battlefield model was a p-kilometer x q-kilometer rectangle. For simplicity the terrain was assumed to be flat and open, so that no geometry of the battlefield was present to block the view of any sensor.

On the foe side, there were *m* foe objects of different types, namely, foe tanks and foe scout vehicles. We will refer to scout vehicles simply as scouts. The scouts move around to gather information about friendly tanks, try to avoid being hit by friendly tanks, and at the same time try to destroy friendly tanks. The differences assumed between tanks and scouts were as follows.

(1) A scout carries a missile launcher with a high hit probability if the scout is within 1km range of a target.
(2) A scout is more maneuverable than a tank.

We assume that there are *n* stationary tanks on the friendly side. We could handle it if they maneuvered, but we considered only stationary friendly tanks (pillboxes) to simplify the implementation. The sensor system of each friendly tank covers a fan-shaped sector. We require that the part of the battlefield beyond the 300 meter line (from the bottom line) be fully covered by the three friendly tanks in normal situations. The friendly tanks are controlled by a multiagent control system intended to search for and to engage the foe objects. For the multiagent control system to make decisions, the friendly tanks send the information about the foe objects gathered by their sensors to the multiagent control system.

5. The Decision Rules

Each agent representing a platoon member has a set of rules made by a higher order commander. To make decisions based upon these rules, the agent is simulated as a nondeterministic automaton. The human operator plays an important role in making decisions in the following situations.

(a) Several choices are available.
(b) The system is in a fuzzy state.
(c) There is no choice at all.

Rule-based real time decision systems have been studied for twenty five years and have not been very successful in practice. One difficulty is the requirement of massive real time rule processing. Another difficulty is distribution of inference tasks among agents. If the rules can't be decomposed into highly separate parts for different agents, the simultaneous inferences required can be very taxing. If a good decision could be made by passing five messages but there are only three messages that can be passed before making a decision, what do we do? A last difficulty is the inability to handle stability questions.

In contrast, in our distributed cost based methodology, we replace rule based decision making by cost based decision making. This eliminates many difficulties associated with human operators, and, at the same time, real time performance is possible because distributing numerical parameters beats chaining through a distributed knowledge base. In this model, the messages which are passed between agents are mostly numerical values which update certain parameters of the cost functions of the agent, and take very little of the capacity of communication lines.

We give samples from the rule base first. We then describe some of the cost functions that we relied on in the next section.

Sample rules for the friendlies:

1. Engage with the closest foe object.

2 If the distance from two foe objects is approximately the same, engage the more dangerous foe object.

3. Scouts are more dangerous than tanks.

4. Among the three modes of the foe objects, the ``engage mode" is the most dangerous. Traverse is less dangerous. Escape is the least dangerous.

5. In general, don't let more than one friendly tank engage with a single foe object. However, in some cases, such as when one foe object is much more dangerous than the rest, more than one friendly can engage the same foe object.

6. If a friendly tank is already assigned a particular foe object, then it has a higher priority of assignment to that same object in the next control command. However, if there is an identified ``emergency", it can be assigned to engage a new foe object even if it the current task is unfinished.

6. Designing Cost Functions

Recall that each agent has its own model of the system. There are three sets of parameters which contribute to decisions of agents. The first set consists of sensor readings. The second set consists of information passed by fellow agents to update the global state of the model. The third consists of higher order commander policies, which can change the decision making entirely. For our example we denote these parameters by r's, λ's and m's.

For a single agent in the platoon, a particular friendly tank, its action decision is based on numerical formulas W_1, W_2, W_3, W_4 and W_5 which represent the five foe objects. The three sets of parameters are as follows. In this description the subscript I corresponding W_i is omitted.

1. Let r be the distance between the agent and the i-th foe object.

2. Let λ_A take values 0, 1, 2 corresponding to the number of other agents currently targeting the underlined foe object.

   Let λ_{ty} represent the foe type, taking values 0, 1 corresponding to being a scout or a tank.

   Let λ_{md} represent the foe mode and take values 0, 1, 2 corresponding to "Engage", "Traverse" and "Escape".

   Let λ_v represent the velocity.

   Let λ_{aa} represent angular acceleration.

   Let λ_{hit} represent the hit probability.

3. m_A, m_{ty}, m_{md}, m_v, m_{aa}, and m_{hit} are corresponding "weights" of the parameter λ's. Furthermore, let m_{wi} be the "weight" which is assigned to the i-th cost function. By adjusting these weights, the higher agent has control of decision making for the platoon.

6.1 Control Orders

The agent computes a command every second. A control order is of the form of a tuple (i_0, T_c), where i_0 is the index of the foe object which is to be engaged with. Here T_c is a real number representing the time allowed to destroy the foe object.

6.2 Prototype Cost Functions

All parameters in the following formula have subscript i.

$$W_i = m_{wi}T_i(t)(\lambda_A m_A r + \lambda_{ty} m_{ty} r + \lambda_{md} m_{md} r$$

$$+ r + |\lambda_v| m_v r + |\lambda_{aa}| m_{aa} r_i + \frac{m_{hit}}{\lambda_{hit}}).$$

The foe object that is to be engaged is computed as:

$$I_0: W_{i0} = min\{W_1, W_2, W_3, W_4, W_5\}.$$

If a tie occurs, choose the one that will require the least gun barrel motion.

$T_i(t)$ is a function used to stabilize the decision method.

Let $T_i(t) = 1$ if the I-th foe is not the target according to the last command of this agent. If i is the target assigned last time, then $T_i(t)$ is a function introduced to stablize the decision. It decreases linearly as time elapses. In this particular model, we took T_c to be a fixed number when the target is new. If the decision is to target the same foe as was targeted in the last command, then T_c is to be the time left according to the last command, say $T_C = T_c - 1$, since one second has passed since the issue of the last command.

These are some examples of cost functions used. In a more detailed mode, one could include new terms representing facts such as the position of the gun barrel, network performance, etc.

The choice of the target as that of least value is an optimization procedure. The weights chosen are crucial to the decision. By modifying these weights, the platoon leader has substantial control of the decision making. We omit discussion of the relation between local optimization and global optimization for lack of space.

7. The Software Implementation

Our hierarchical multiple agent command model for automated forces was designed to minimize communication costs and distribute computation as much as possible to individual agents. Each agent can be implementated on a work station, or as a group of processes, or on a PC. The agent network is implemented by networked computers. We chose the TCP/IP protocol to implement the communications

among agents. It may be better to build a communication kernel protocol directly.

We describe the software implementation briefly.

7.1 The Target Designation Module

The agent target designation module assesses the environmental threat and designates targets for the tank associated with agent. The multiagent controller is formulated as a distributed optimization problem with specially designed cost functions.

In our simplified model, the parameters have deterministic values. The model is not event driven, that is, a command is computed every second based on the information available at the time.

7.2 The Data Interpretation Module

Because of the uncertainty of sensor data and possible time delay in message passing, one has to process raw data with some mathematical tools, i.e. probabilistic tools. The data interpretation module interprets the measurements and coordination messages, and resolves inconsistencies in data. The measurements and messages processed by the data interpretation module produce the parameters for foe mode, foe type and hit probability. Based on some task decomposition rules, each agent is responsible for the parameters of a subset of foes. The parameters then are passed to other agents.

7.3 The Sensor Management Module

The sensor management submodule schedules the area to be covered by the sensor system. The sensor of each friendly tank covers only a sector of battle field. However, the sectors are subject to change in order to guarantee optimal performace of the friendly sensor system. For example, if too many foe objects fall into the sector of one particular sensor, the system may want to change the covered sectors of the tanks.

7.4 The Communication Module

The communication module is responsible for sending coordination messages to fellow agents. For each agent, the only way to get access to global information is by message–passing. This is controlled by the communication module. For communication among a platoon group, we used star shaped network topology. That is, we implemented a virtual gateway to handle all messages. In this gateway, there were buffers for each agent containing dynamically sorted messages waiting to be transferred. The messages from the platoon leader agent also go through this gateway. As one can see from the last section, the weights of the corresponding parameter determine the relative importance of the parameter to the decision. Because we use a real time default scheme in computing decisions, an algorithm for dynamically sorting messages in the buffer according to importance was implemented.

8. Conclusions

We introduced a multiple-agent hierarchical command system for automated forces based on distibuted optimization of local cost functions.

A sample cost-based decision system for a model of the problem of engaging multiple targets with multiple weapons platforms was implemented at Picitinny Arsenal with the cooperation of Norman Coleman and Wolf Kohn [14]. A software demonstration of the computer generated system controlled by cost based distributed controllers is available at Picatinny Arsenal and at ORA corporation.

While the model which was implemented was not at the same level of detail as those being experimentally integrated to build CIS's, the demonstration did establish the feasibility of applying a declarative approch to integration of heterogeneous battlefield models. The high-level logic of assigning sectors of fire and performing target detection, identification and assignment was declaratively integrated with the low-level differential equation models of target motion and the engagement process. This approach supports easy change of the battle drill logic as well as the easy inclusion of more accurate nonlinear differential equation models of weapon system characteristics.

9. Acknowledgements

Research supported by Sagent Corp, ORA Corp., DARPA-US ARMY AMCCOM (Picatinny Arsenal, N.J .) contract DAAA21-92-C-0013, and U. S. Army Research Office contract DAAL03-91-C-0027.}

10. References

[1] K. Birman, The Process Group Approach to Reliable Distributed Computing, CS Tech Report 91-1216, Cornell University, July 1991, Revised September 1992.

[2] J. N. Crossley, J. B. Remmel, R. A. Shore, and Moss E. Sweedler, *Logical Methods: In honor of Anil*

Nerode's Sixtieth Birthday, Progress in Computer Science and Applied Logic vol. 12, Birkhauser, Boston, 1993.

[3] I. Ekeland, *Infinite Dimensional Optimization and Convexity*, University of Chicago Lecture Notes in Mathematics, University of Chicago Press, 1983.

[4] X. Ge, W. Kohn, and A. Nerode, Algorithms for Chattering Approximations to Relaxed Controls, (Patriot's Day, April 94, 1994). MSI Technical Report 94–23, Cornell University, April 1994.

[5] X. Ge, A. Nerode, Effective Content of Calculus of Variations I: Semicontinuous Functions and Chattering Lemma, MSI Technical Report, Cornell University, March 1995.

[6] R. L. Grossman, A. Nerode, A. Ravn, and H. Rischel, eds., *Hybrid Systems*, Lecture Notes in Computer Science 736, Springer-Verlag, 1993.

[7] W. Kohn, A Declarative Theory for Rational Controllers, Proc. 27th IEEE CDC, pp. 130-136, 1988.

[8] W. Kohn, Autonomous Space Robots, Proc. AIAA Conf. on Guidance, Navigation, and Control, vol. 1, pp. 382-390, 1988.

[9] W. Kohn, Cruise Missile Mission Planning: A Declarative Control Approach, Boeing Computer Services Technical Report, 1989.

[10] W. Kohn and A. Nerode, An Autonomous Control Theory: An Overview, Proc. IEEE CACSD92, March, 1992.

[11] W. Kohn and A. Nerode, Multiple Agent Autonomous Control Systems, Proc. 31st IEEE CDC in Tucson, pp. 2956-2966.

[12] W. Kohn and A. Nerode, Models for Hybrid Systems: Automata, Topologies, Controllability, Observability, in *Hybrid Systems* [6].

[13] W. Kohn and A. Nerode, Multiple Agent Hybrid Control Architecture, in [6], 1993.

[14] J. Lu, X. Ge, W. Kohn, A. Nerode and N. Coleman, A Semi-Autonomous Multiagent Decision Model for a Battlefield Environment, MSI Technical Report 94-55, Cornell University, Oct. 1994.

[15] A. Nerode and W. Kohn, A Hybrid Systems Architecture, in [2], 1993.

[16] J. James, N. DeClaris, Wolf Kohn, and A. Nerode, Intelligent Integration of Medical Models, Proc. IEEE Conference on Systems, Man, and Cybernetics, San Antonio, pp. 1-6, Oct. 1994.

[17] A. Nerode, J. James, and W. Kohn, Multiple Agent Hybrid Control Architecture: A generic open architecture for incremental construction of reactive planning and scheduling, Intermetrics Corp. Report, June, 1994.

[18] A. Nerode, W. Kohn, J. B. Remmel, and X. Ge, Multiple Agent Hybrid Control, Carrier Manifolds and Chattering Approximations to Optimal Control, Proc. IEEE 33rd CDC, vol. 4, 1994.

[19] *Proceedings of the ARO Workshop on Hybrid Sytems and Distributed Interactive Simulations, Feb 28 - March 1, Research Triangle Park, N.C.*, 1994.

[20] W. Kohn, J. James, A. Nerode, and J. Lu Multiple-Agent Hybrid Control Architecture for the Target Engagement Process (MAHCA-TEP - Version 0.2 of MAHCA-TEP: Technical Background, Simulation Requirements, and Engagement Model), Intermetrics Technical Report, 25 August, 1994.

[21] W. Kohn, J. James, and A. Nerode, Multiple-Agent Hybrid Control Architecture for Distributed Interactive Simulation, in [19].

[22] B. McEnany and H. Marshall, CCTT SAF Functional Analysis, *Proceedings of the Computer Generated Forces Conference*, May, 1994.

[23] COL James E. Shiflett, Memorandum, Subject: Use of Doctrine Based Software Development in the Development of CCTT, Department of the Army, PM CATTS, Nov.4, 1994.

[24] DIS Steering Committee, The DIS Vision A Map to the Future of Distributed Simulation (comment draft), (Margaret Loper, UCF, Chair; Steve Seidensticker, SAIC, DIS Vision Document Coordinator), Oct. 1993.

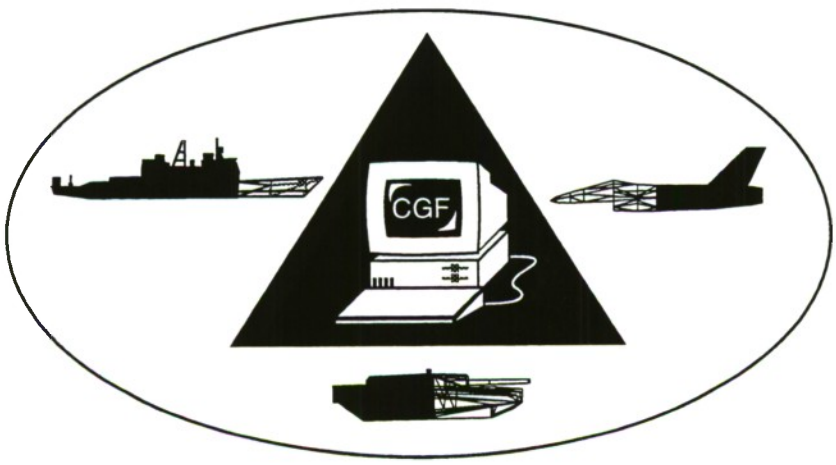
11. Authors' Biographies

Dr. Xiaolin Ge is in the second year of a post-doctoral research associateship at the Mathematical Sciences Institute of Cornell University. His work is sponsored by the Army Research Office and the ARPA DSSA program. He received his Ph.D. degree in Mathematics from the University of Minnesota in 1993. He has co-authored papers with Anil Nerode, John James, Wolf Kohn, J. B. Remmel, and J. Lu in Hybrid Systems and Distributed Control. He is also the author of ten papers, including some with Anil Nerode and some with the late Ian Richards, in mathematical logic, algorithmic algebra, functional analysis, and nonlinear optimal control. His current research program is devoted to the design and implementation of distributed multiple agent hybrid control systems.

Dr. John James received a B.S. from the United States Military Academy in 1967, an M.S.E.E. from the

University of California, Berkeley in 1973 and a Ph.D. in Electrical Engineering from Rensselaer Polytechnic Institute in 1986. He is participating in the development of tools to support modeling, simulation, design and implementation of control systems, particularly intelligent control systems. Dr. James has been involved in building analysis and design tools for control system design for over twenty years, including three years as Chairman of the IEEE technical committee for Computer-Aided Control System Design. As Director of the U.S. Army Training and Doctrine Command (TRADOC) Artificial Intelligence (AI) Center he directed the development of over 30 knowledge-based decision support systems for the TRADOC staff at Fort Monroe, Virginia. TRADOC plays a key role for the Army at two points in the life-cycle of acquiring weapon systems: (1) TRADOC writes the requirements documents which start the acquisition process and (2) TRADOC writes the critical operational issues and criteria documents which are used to evaluate whether the item as built meets the user's needs as stated. The TRADOC AI Center supported activities at TRADOC schools and installations which included: the Future Battle Laboratory at Fort Leavenworth, Kansas which developed requirements for Army Corps-level-and-below command and control, the Technology Assessment Center at Fort Huachuca which developed requirements for Intelligence and Electronic Warfare, the Technology Assessment Center at Fort Gordon which developed requirements for communications, and the TRADOC Analysis Command at Fort Leavenworth which developed force-on-force simulation software for corps-level-and-below operational analysis studies. Active in the Institute of Electrical and Electronics Engineers (IEEE), he was co-organizer and co-chaired the first invited session on hybrid systems technology at the 1992 IEEE Conference on Decision and Control. He was also general chairman of the 1992 IEEE Symposium on Computer-Aided Control System Design. He is currently President of Sagent Corporation.

Anil Nerode is Goldwin Smith Professor of Mathematics and Computer Science and Director of the Mathematical Sciences Institute at Cornell University, an Army Research Office Center of Excellence. He received his Ph.D. in Mathematics at the University of Chicago in 1956 under Saunders MacLane, was an NSF postdoctoral fellow with Kurt Godel at the Institute for Advanced Study in Princeton and then with Alfred Tarski at the University of California at Berkeley. He is the author of a hundred and fifty papers and books in mathematical logic, algebra, recursive functions, automata, recursive and polynomial time algebra, concurrency and distributed systems, non-monotonic reasoning systems, logic programming, hybrid systems, and distributed control. He has produced thirty five Ph.D.'s who hold senior positions in mathematics and computer science departments nationwide. He was an editor of the Journal of Symbolic Logic for sixteen years, has been an editor of Annals of Pure and Applied Logic for fifteen years, is the associate editor of the Annals of Math and AI, and is also editor of many other journals in mathematics, applied mathematics, computer science, AI, and modelling and simulation. He has been a military consultant to all the armed forces in design of weapon systems over a period of forty-one years. He is currently a Vice President of the American Mathematical Society and a member of the Committee on Mathematics of NAS/NRC.


CGF

Context-based Representation of Intelligent Behavior in Simulated Opponents

Avelino J. Gonzalez
Electrical and Computer Engineering Dept.
University of Central Florida
Orlando, FL 32816-2450
ajg@engr.ucf.edu

Robert Ahlers
Naval Air Warfare Center
Training Systems Division
Orlando, FL 32826

Abstract

This article describes and evaluates a concise, yet rich representation paradigm that could effectively and efficiently be used to model the intelligent behavior of opponents in a simulation-based tactical training system. This feature would be quite useful in the training process for two reasons: 1) the trainee would face a realistic enemy who is knowledgeable about tactics in the domain of interest and, 2) the instructor would not be burdened with playing the part of the enemy in those training systems where this is commonly done.

The representation paradigm proposed is based on the idea that applicable tactical knowledge is highly dependent upon the situation being faced by the decision maker (i.e., the context). A combination of script-like structures and pattern-matching rules in an object-oriented environment could serve to hold all knowledge pertinent to the context present at a specific time. This paradigm has been preliminarily tested in a prototype system that incorporates the knowledge of a submarine tactical officer on a patrol mission. Evaluation of the prototype shows that the context-based paradigm promises to meet the desired levels of conciseness and effectiveness required for the task.

1. Introduction

The use of autonomous and intelligent simulated adversaries in a training simulation can provide a more realistic experience to the trainees than would be presented otherwise. This would be especially true for tactical training simulators if an intelligent simulated adversary could be made to react and counter the trainee's tactics in a realistic fashion. Intelligent simulated adversaries will be henceforth referred to in this article as *Autonomous Intelligent Platforms* (or AIP's). They can be defined as instance representations of military platforms (i.e., submarine, destroyer, tank, helicopter, fighter aircraft) in a training simulation which behave as would a real platform in actual battle from a tactical standpoint.

Tactical knowledge is required in order to endow AIP's with the ability to act, not only intelligently, but also realistically, in light of a trainee's actions. In general, tactical knowledge can be said to address time-stressed tasks which require 1) assessment of the situation at hand, 2) selection of a plan to most properly address the present situation, and 3) execution of that plan [Thorndike, 1984]. It is how this knowledge is represented and manipulated that is the focus of this research project.

Most tactical tasks in the military consist of a pre-defined set of actions which are embarked upon after a certain situation has been recognized. The situation could be a mission, a set of orders, or merely a reflection of a specific set of battle conditions at the moment. The problem faced, therefore, is two-fold: 1) how to recognize the present situation (referred to as situational awareness), and 2) what to do when the situation is recognized (referred to as implementation of actionable information).

The basis of our approach to the problem of concisely and effectively representing AIPs is based on the following hypotheses:

1) Tactical experts are proficient at their task by recognizing and treating only the

key features of the situation, and abstracting these for use as the premises for the general knowledge. Thus, they only use a small, but important portion of the available inputs. An example of this is the tactical exercise of driving an automobile.

The driver of an automobile is generally bombarded with a multitude of inputs when driving: audio inputs such as engine noise, road noise, the radio, conversation with passengers, etc.; visual inputs such as the instruments, other automobiles, the surrounding scenery, pedestrians, etc.; tactile inputs such as vibrations of the car, the position of the steering wheel, the gear shifter, the clutch, etc.. These inputs are cognitively handled rather easily by the driver when they are all in the normal or expected range. However, if one of these should deviate from normal, such as the abnormal noise and vibrations resulting from a tire blowout, the driver will immediately focus on these inputs in order to recognize the present situation as a blowout, while ignoring all the other inputs being received.

2) There is only a limited number of things that can realistically take place in any situation. Using the example above, it would not be expected that a tire blowout take place while waiting at a stop light. This can be used to advantage to prune the search space of the problem, since there is no need to consider a blowout while waiting at a stoplight. Getting rear-ended, on the other hand, is a much more likely proposition.

3) The presence of a new situation will generally require alteration of the present course of action to some degree. For example, the recognition of a blowout at highway speeds will cause the driver to coast to a stop while maintaining a hard grip on the steering wheel, and directing the car towards the shoulder of the road. Thus, the context changed from one of normal driving, to one of blowout, with its attendant course of further action. This context remains in effect until the car comes to a complete stop, at which point another situation will be recognized and acted upon (i.e., get out of car, inspect tire, change tire etc.).

The work described here is based on the idea that by associating the possible situations and corresponding actions to specific contexts, the identification of a situation is simplified because only a subset of all possible situations are applicable under the active context. The work also addresses what course of action to follow when a situation is correctly recognized.

2. General Description Of Context-based Representation Paradigm

Our approach to implementing the concepts described above lies in the use of *Scripts*. A script is a knowledge representation paradigm developed by Roger Schank at Yale University [Schank, 1977] which attempts to capture the actions, objects, persons, and concepts that may be related within a given context. For example, a restaurant script will be composed of all the actions which are typically part of going to a restaurant, such as reading the menu, ordering the meal, eating it, paying the bill, etc. A restaurant script also contains props, objects which are typical to a restaurant scene from the customer's standpoint, (e.g., tables, chairs, menus, food, eating utensils, napkins, salad bars, etc.) as well as actors (i.e., waiters, hostesses, chefs, busboys, etc.). The actions involved are only those typical of the restaurant experience. It would not be normally expected, therefore, that the customer wash her car at the restaurant.

This concept can be easily extended to military tactics, where a script can be used to express the set of steps (at either a high or low level) that are necessary to carry out the action required by the present situation. Within the context of a mission, there is a limited number of things that are generally expected in terms of actions to carry out and the expectations in regards to the possible situations. It would be quite difficult to represent all this knowledge using rules alone. Thus, the basis for this research project is to use scripts (in addition to objects as well as a minimal number of rules) as the knowledge representation paradigm for a set of AIP's. For lack of a better name, this representation and reasoning paradigm will be referred to as *Context-based Reasoning* (CxBR).

The approach proposed here is based on the following assumptions:

1) Life for an AIP is a continuous and dynamic decision making process. Decisions are heavily influenced by a never-ending sequence of contexts, each of which, when active, regulates the behavior of the AIP as well as provide an expectation for the future. Active contexts change not only in response to external events or circumstances, but also as a result of actions taken by the decision maker (the AIP itself). A context can be likened to a situation that has been recognized, and which has a prescribed set of procedures that must be carried out, either sequentially, methodically, or arbitrarily. One example of a context would be driving an automobile on an interstate highway at normal cruising speeds. The behavior of an AIP in that situation is controlled by the context that is active for it at the time.

2) The active context may not be the same for all AIP's at the same time. This is reasonable to expect, since each may have a different mission, different sensor inputs, different capabilities, a different physical location, etc.

3) At least one specific context or set of contexts is always active. More than one context can be valid, but only one may be active. Furthermore, all valid contexts must be compatible. Thus, one context must be the central focus of attention. For example, using the case of traveling in an automobile, the driver may be cruising on an interstate normally at the speed limit, a situation which may be characterized as **normal-highway-driving**. Moreover, he/she may also be hungry which can be as a situation labeled **driver-hungry**. If he/she is more anxious to arrive at the destination than willing to satisfy the hunger, then the first situation will dictate the action being undertaken. Thus, the active context would be **normal-highway-driving**. Otherwise, a **driver-hungry** context will be the active one and the action will shift towards finding a place to eat.

4) Contexts are represented temporally as intervals of time rather than time points. Contexts can be considered to be transitions to reach a goal (look for a place to eat), or they can be a goal in themselves (eating).

5) Goals can be time points, but only to serve as transitions to other contexts. For example, arrival at a destination can be defined as a goal of **normal-highway-driving** and it can be represented as a time point, but it is not a context in its own right, only a transition to another context (e.g., **being-there**). This process may go on until the mission ends.

6) Only a limited number of things can take place in any single context. A situation, therefore, by its very nature, will limit the number of other situations that can realistically follow. Using as an example the domain of submarine warfare, it would not be expected that the submarine would be attacked in its own home port. This can be used to advantage to prune the search space of the problem, since there is no need to watch out for a torpedo attack while waiting to be resupplied at port. If unexpected situations do take place, that introduces the element of surprise into the AIP's behavior, which is highly consistent with the real world.

7) Certain cues exist which will indicate that a transition to another context is desirable. This makes use of the hypothesis that experts look for certain few specific cues which that will indicate a new situation (e.g., the vibration on the steering wheel upon a tire blowout.)

8) The presence of a new context will alter the present course of action and/or the applicable expectations to some degree. For example, the recognition of a blowout at highway speeds will cause the driver to attempt to coast to a stop while maintaining a firm grip on the steering wheel, and directing the car towards the shoulder of the road. Thus, the context changed from **normal-highway-driving**, to one of **tire-blowout**, with its attendant requisite action. This context remains in effect until the car comes to a complete and safe stop (the goal), at which point another context will be recognized and acted upon (e.g., **fix-tire**).

By associating the potential contexts and corresponding actions to specific situations, the identification of a situation can be simplified because only a subset of all possible situations is applicable under the active context. This context-based approach also easily addresses what course of action to take when a situation is recognized.

This is in some ways similar to a system proposed by Thorndike [1984] called "Context Template-driven SAFOR". However, the latter appears to implement AIP's that are intelligent in a much higher level in that they can only respond to orders from higher command, or requests from subordinates. They do not appear to be able to perceive the environment independently. Moreover, the transition from one context to another appears to be significantly more rigid that what is being proposed here, which may lead to unintelligent decisions. Lastly, the system seems to not provide the capability to its AIP's to plan, a significant disadvantage.

Czejdo and Eick [Czejdo, 1993] present an environment in which addresses the problem of large-scale knowledge management. Called the Tanguy Knowledge Base Management System, it integrates rules, objects and database features in order to take advantage of their respective features.

Dreyfus and Dreyfus [Dreyfus, 1986] take exception to the idea of using contexts to simulate human intelligence in computers. They correctly point out that there can exist many contexts in the course of human life, and to attempt to account for all of them is a hopeless task. Nevertheless, their argument arises from the standpoint of refuting the claims that computers can be intelligent in the same way as humans, in all aspects of human intelligence. The objective of this work is less ambitious in scope, since the AIP's do not have to have the breadth of knowledge possessed by humans in order to appear intelligent in a training simulation. Rather, they only must appear to behave as a human enemy would in a very specific and narrow domain (i.e., submarine warfare, tank warfare). In fact, this may mean that they should not display optimal behavior, as that is not always typical under wartime stress. It is our belief that applying context-based reasoning as described below presents a highly effective and efficient methodology for imparting sufficient intelligence to AIP's so as to achieve their objective in a training simulator.

2.1 Representation of Contexts

In CxBR, contexts are the most important representational item. Much knowledge about how the AIP should behave, as well as to what other contexts it can transition is stored in the context objects themselves. There are three levels of contexts that can be represented, and they are ordered hierarchically. These are 1) the mission context, 2) the major contexts and 3) the sub-contexts. These will be described below.

2.1.1 Mission Contexts

A mission-context (simply referred to as a mission) is an overall definition of the objectives of the scenario. It defines the objectives as well as the constraints of the operation. The mission can also define the things to avoid during the mission. Examples of missions in the domain of submarine warfare are SEARCH-AND-DESTROY enemy submarines, MINING a harbor or choke point, GATE-KEEPING, BATTLE-GROUP-ESCORT, ANTI-SURFACE-OPERATION, SPECIAL-OPERATIONS, and others. A mission can define the types of lower level contexts which may be necessary during the execution of this mission. A mission context can also describe the political environment under which the mission is to be carried out. For example, if a hot war is in effect, then the rules of engagement will surely be different than if a cold war is in effect. No more than one mission will be active at any one time, and missions are mutually exclusive. So, a new mission would have to bump an existing mission from active status. In practice, however, there would be little need to change missions during the course of a training session.

The mission defines the constraints as well as the Major-Contexts therein. It is a class definition in an object-oriented environment and contains the following attributes:

> **Constraints:** This attribute lists all the constraints that are imposed on the AIP during this mission. Some of these could be: withhold fire unless fired upon, any limitations placed upon the submarine's performance characteristics, etc.
>
> **Avoid:** This attribute describes anything that must be avoided at all times throughout the training scenario. One obvious one is destruction-of-self, but there may be others such as avoid counter-detection at all costs, etc.

Major-Contexts: This attributc lists the Major-Contexts present in the mission. For example, for a patrol mission (callcd SEARCH-AND-TRACK), the major aetions for the AIP will be getting to its assigned sector, searching the sector in an appropriate fashion, tracking an enemy eontact if one is found, and breaking contact to return home when certain parameters are fulfilled.

2.1.2 Major-Contexts

Major-Contexts are the main focus of the Context-based reasoning and representational paradigm. They contain all the necessary information to operate the AIP, as well as to determine when the aetive context should be deactivated and another one put in its place.

A major-context (or simply context for short) is a tactical operation undertaken as part of the mission in order to assist in achieving the goals set forth. One context is always in control of thc AIP, and contcxts are, also by definition, mutually exclusive of eaeh other. Unlike the mission, however, contexts are normally activated and deactivated many times throughout the course of a training session. A context is activated by retraeting from the faet base the fact that identifies the active status of the eurrcnt eontext, and calling the initialization proeedure of the newly activated context. The latter will assert into the fact base a new fact identifying the new context as the active one. This will allow the monitoring rule(s) that pertains to this context to become aetive and fire periodically. Furthermore, the initialization message will also modify any parameters that so require modification, sueh as possibly the AIP's heading, speed, depth, etc. In some missions, the sequence of some of the contexts will be known a-priori. Although some contexts may become invalid through the simple passage of time, this is not eommon. In most cases, the context will cease to be applicable due to either an action taking place or thc completion of its task. Therefore, contexts are generally assumed to be aetive indefinitely, until bumped from active status by another context.

Eaeh context is defined as a class in an objeet oriented environment, and possess the following attributes:

Initializer: Referenees the namc of the message-handler whieh is executed whenever the context/sub-context is first aetivated to initialize all required variables.

Objective: The objective slot puts a message as to what the objective of the context/sub-context is. The objective is in general terms and it referenees a frame that has some attributes that are the goal of this context/sub-context.

Compatible-next-major-context: This attribute lists those contexts to which transition from the current context is acceptable.

Compatible-sub-context: This attribute is a list of all sub-contexts whieh are compatible with the current context. For example, it would not be advisable to put an automobile in cruise-control when a blowout has taken place. Thus, the cruise-control context would not appear on that list.

Additionally, some eontexts will have slots that are specific only to them, and deal with universal variables that need to be known throughout the entire simulation. For example, the attack eontext will have a slot defining the target of the attack and another defining the number of weapons used. Likewise, the under-attack context will have a slot defining the aggressor (source of the weapons bearing down on the AIP).

There may be other attributes added to the class definition for context/sub-contexts in the future. One particularly desirable would be a further refinement of the compatibility aspect by providing a numerical weight to each context to decide which one would be more desirable in the case where more than one would be acceptable given the current situation. This competing context concept is further described later in this article.

2.1.3 Sub-Contexts

Sub-contexts are lower level tactical procedures whieh are not critieal in and of themselves to reaching the mission objectives. They are typically of temporally short duration. Sub-contexts are at this time mutually-exclusive with one another, but can be eompatible, and thus co-exist, with the contexts. It is expected, however, that in the future, compatible sub-contexts may co-exist with one another on active

status as long as they control different variables. In cases of incompatibility, a sub-context will not be activated when an incompatible context is active. Likewise, when a new context is activated while an incompatible sub-context is active, the sub-context is immediately "short-circuited". In any case, however, the contexts always take precedence. It is not necessary for one sub-context to be active at all times as is the case with contexts. When no sub-context is active, the sub-context is said to be "none".

The attributes of a sub-context objects are quite similar to those of a context, and thus will not be described further.

2.2 Situation Assessment and Transitioning Between Contexts

One of the foundations of the CxBR approach is that by knowing what the AIP is doing at any one time, it can know what to expect. This greatly facilitates the task of situational assessment. One example in the automobile driving domain is that when on an interstate highway, one does not have to be concerned with traffic crossing the roadway, as there are no intersections per se. When driving on city streets, however, one of the most dangerous situations is when the automobile approaches intersections, and thus, a driver has to be especially aware of them. The situational awareness function in the existing prototype is done by simply looking for parameter values that indicate that a change in context is warranted. This is a rather simple, yet quite effective means of doing situational assessment under CxBR.

The basic recognition of the situation is done through pattern-matching rules. While this might not seem to be a concise way of carrying this out, the use of the active-context and/or active-sub-context pattern in the rule premise will significantly limit the solution space of the search as was described in the previous section. Rules will have a pattern in their premises that indicates the active context to which they are applicable. Only when there is a fact in the factbase indicating the active status of the appropriate context will these rules be "active" and capable of being executed.

The transition among the various contexts is a critical issue in CxBR. This approach is based on the use of monitoring rules. Each major-context/sub-context will have at least one of these. These rules will fire continuously (every simulation cycle) as long as its parent context is active. In its right hand side, the rule will have a conditional statement(s) that will monitor the parameters which are relevant to the continuation of the context. Examples of these parameters are: whether the enemy contact has been detected, whether it is moving towards or away from the AIP, whether it is within firing range, etc. Once these parameters are satisfied and a change of context/sub-context is indicated, the rule will retract the fact that advertises the active context/sub-context. This will prohibit these monitoring rules from firing any longer. The rule will have the transition information embedded, so that it will call the initializing message for the new context/sub-context. The initializer will set the revised parameters on the AIP as may be required, and post into the factbase the new facts that announce the newly-activated context/sub-context. This will allow the monitoring rule for that context to begin firing.

In some cases, the transition to a new context/sub-context would be a result of an "external" message (e.g., a communication from fleet command). Examples of such would be a message to return home, or go to periscope depth to receive or transmit a more detailed communication. This external message is represented as a fact posted on the fact base which has a "prompt" indication (i.e., (communication prompt)). In such circumstances, an additional rule is required, which fires only once, to retract the current context/sub-context fact (as well as the prompt fact) and to invoke the initializing message that activates a new context/sub-context.

Of course, there are also universal monitoring rules which are not tied into any one context/sub-context. These rules search for situations which could occur under any context, such as being fired upon by an aggressor, or the detection of an enemy contact.

A more thorough discussion of CxBR is included in [Gonzalez, 1993; 1994].

3. Implementation and Evaluation of the CXBR Approach

In order to properly evaluate the ideas set forth in this article, two prototypes were developed and tested. Prototype #1 was developed in the submarine warfare domain, while prototype #2 was in automobile driving. Both of these will be discussed in this article,

but the first prototype is more comprehensive in nature and will thus be discussed more thoroughly.

The two prototypes were implemented in CLIPS 5.1. This environment proved to be a good framework for the task, but certainly not ideal. Memory limitations of the DOS-based CLIPS version, its inability to match facts in the factbase with patterns anywhere but within rule premises, and its basic inadequacy as a simulation tool often made it cumbersome to use.

3.1 Submarine Warfare Prototype

While the implementation of submarine tactics was not per se an objective of the current investigation, it became clear that to design an appropriate architecture and develop a prototype that verified the use of that architecture, a limited set of submarine warfare tactics had to be defined. The first prototype was thus built to evaluate the behavior of the AIP in a SEARCH-AND-TRACK mission, in the presence of one enemy submarine (called *ownsub*). In order to make it self-contained, the prototype incorporated its own simulation of the submarine warfare environment, which took up considerable computing resources.

All missions, major-contexts and sub-contexts were represented as classes in the CLIPS Object Oriented Language (COOL), as were the SUBMARINE classes and the weapon classes. Monitoring rules, of course, were implemented as CLIPS rules. Initialization messages were implemented as message-handlers in COOL. The central manager was composed of the main loop which contained all the procedures that were to be repeated every simulation cycle, and a number of other functions which carried out calculations such as distances, bearings, etc., when required. The output was in the form of a text report which, every five seconds, listed the x and y positions as well as the depth, of all submarines and weapons involved in the scenario.

Interruptions could be made to the simulation to introduce control of the AIP by the instructor. Interactions permitted with the AIP through this interruption mechanism included orders to attack, communication prompts, baffle-clearing prompts, and return home orders. Interactions with other simulated submarines (called *ownsub*, and driven by the student being trained) included changing its heading, depth, speed, and firing its weapons.

Upon satisfaction that the full Prototype #1 operated correctly, a version of it was built that could be interfaced with an external graphical simulation. This would allow the investigators to evaluate the feasibility of incorporating this technique within existing simulator training systems, a critical step in verifying its usefulness. The simulation employed for this purpose was the *Intelligent Platform Modeling System* (IPMS), a testbed being developed at the Naval Air Warfare Center, Training Systems Division in Orlando, FL.

The externally-interfaced version of Prototype #1 was developed by stripping off all the code from the full prototype which served to support its built-in simulation. A networked interface was used as the means of communication between the CLIPS-based AIP model and the DOS-based IPMS simulation. In order to resolve the memory limitations of the DOS-based CLIPS 5.1, a UNIX-based version was employed, running on a Silicon Graphics workstation.

The AIP (also called *opsub*) implemented in Prototype #1 was found to behave tactically correctly from a qualitative point of view when subjected to several different situations. The situations to which opsub was subjected included transiting to its designated sector using a sprint-and-drift tactic, searching the sector for enemy activity, maneuvering into position to track enemy contacts, tracking such contacts, clearing its baffles, getting into position to attack the enemy attacking the enemy when externally ordered, (in the spirit of a reconnaissance mission), and evading enemy attacks. Opsub was placed in these situations through the control of the location, bearing, speed, depth and weapons of ownsub.

The externally-interfaced prototype was able to transit to the sector using a sprint-and-drift maneuver, and carry out a baffle-clearing tactic while doing a search of the sector. It was additionally capable of searching the sector, detecting the enemy (ownsub), maneuvering into position to track ownsub, and breaking contact to return home when told to do so. The prototype also performed an approach to fire its weapons. However, due to the inability of the IPMS to model weapons in the water, evaluation of attacking and evading attacks was not possible.

The qualitative evaluations of the two versions of Prototype #1 described in this sections allows us to

conclude that: 1) the CxBR paradigm can be used to accurately represent the tactical behavior of an AIP from a qualitative standpoint, and 2) the paradigm has been shown to be compatible with a distributed simulation environment.

Conclusion #1 is an essential one, since inability to be used to represent tactical behavior would invalidate the CxBR paradigm without the need for any further evaluation. Conclusion #2 is significant from a usefulness standpoint if AIP's are to be retrofitted to existing simulators. Moreover, it is also important in light of the U. S. Army's interest in distributed interactive simulations.

3.2 Automobile Driving Prototype

Qualitative success in the performance of the AIP prototype does not provide a complete picture of the viability of the CxBR technique. Furthermore, conciseness can only be unequivocally judged by comparing a CxBR prototype with an equivalent purely rule-based implementation of the knowledge and capabilities exhibited by a CxBR prototype. This was accomplished and the evaluation is described in the section that follows.

The scope of the automobile driving prototype was more modest than that of Prototype #1. This prototype used an automobile simulator system [Klee, 1991] and implemented a short scenario where the AIP automobile (labelled *student car*) is cruising on a two-lane road and approaching a curve near an intersection where another car (labelled *simulation car*) is waiting to turn left into the road ahead of it. To complicate matters, a small truck (*simulator van*) is coming around the bend in the opposite direction. Figure 1 graphically depicts a bird's eye view of the scenario faced by the AIP. The AIP is tasked with avoiding a collision with both the car and the van, as the car attempts to cut in between the AIP and the approaching van. The scenario was varied by running various tests with different "release distances" for the car and the van. This meant that the distance available for the AIP to maneuver ranged from one where no real danger was present, to one where a collision was physically inevitable. The courses of action available to the AIP were to: 1) slow down (possibly through the application of brakes) in order to maintain a distance between itself and the simulation car; 2) brake and swerve to the left if there is sufficient distance to avoid hitting the oncoming van; and 3) swerve to the right (off the road) if there isn't sufficient distance.

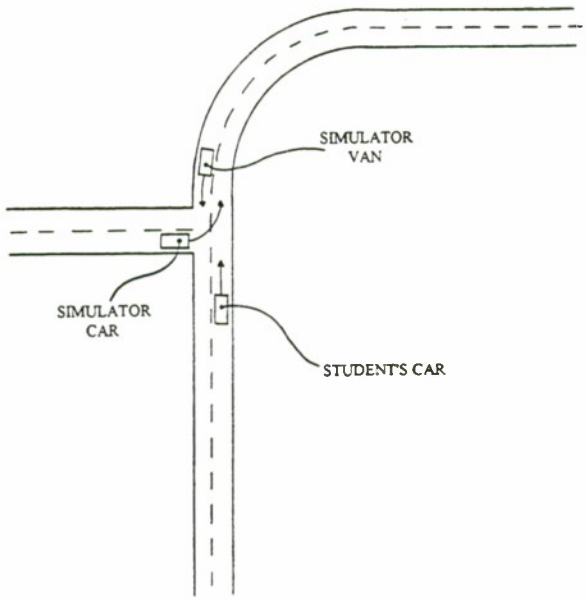


Figure 1 - Bird's Eye View of Simulated Scenario

Criteria of comparison	CxBR	Rules-only
Execution time avg. (sec)	124.1	126.91
CLIPS elements	53	43

Table 1 - Summary of Quantitative Evaluation of CxBR

The tactics used by the AIP to accomplish this were represented through contexts as described earlier in this article. At the same time, a rules-only representation was also implemented for the purpose of comparison. Both implementations were done in

CLIPS 5.1 to make the comparison as straightforward as possible. The parameters compared were 1) the number of CLIPS elements used by each implementation and, 2) the time of execution required by each implementation. The results are shown in Table 1 above. A full description of the evaluation procedure is found in [Brown, 1994].

The execution times for the CxBR implementation were 2.26% better than those for the pure rule-based alternative. Although this difference is nearly insignificant, it does demonstrate that the CxBr is more efficient. This difference is expected to become more pronounced as the size and scope of the domain expands.

In regards to the number of CLIPS elements, the rule-based alternative was actually more concise. Once again, the limited scope of the prototype skews the results, since the CxBR alternative requires a "fixed overhead" in CLIPS elements in order to adequately represent the tactical knowledge. This overhead is in the form of the classes for each context or sub-context defined for the tactic, and the message handlers involved with each class instance. As the situation becomes more complex, this overhead becomes a smaller part of the total knowledge, while the rules required for pure rule-based reasoning become more numerous to account for all the possibilities.

4. Summary and Conclusions

The use of contexts to represent and reason about tactical knowledge has the advantage of encapsulating all facets of such knowledge as it applies to a small slice of the entire domain knowledge. By modularizing the knowledge in such a way, and by explicitly expressing the relationships between the various contexts such that the number of possible transitions between contexts are inherently limited, efficiencies can be implemented. These efficiencies are in terms of economy of knowledge as well as in efficiency of execution of the system.

This article describes the concept of Context-based Reasoning as well as two prototypic implementations of these concepts in order to evaluate its effectiveness in achieving the efficiencies expected. The prototypes were successful in achieving the objectives of the investigation. Nevertheless, areas of further work were discovered where the present system is deficient, namely in how to deal with time, either as historical information or as in planning the next move. Basically, the prototypes are reactionary in nature, as their planning capabilities are rather limited. Nevertheless, from a conceptual standpoint, planning is quite consistent with the general CxBR approach, and such a capability will be featured in future versions of the prototypes.

5. Bibliography

[Brown, 1994] Brown, J. C., "Application and Evaluation of the Context-based Reasoning Paradigm", Master's Thesis, Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, July, 1994.

[Czejdo, 1993] Czejdo, B. and Eick, C. F., "Integrating Sets, Rules, and Data in an Object-Oriented Environment", *IEEE Expert*, February, 1993, pp. 59-66.

[Dreyfus, 1986] Dreyfus, H. L., and Dreyfus, S. E., *Mind over Machine*, New York: MacMillan/ The Free Press, 1986.

[Gonzalez, 1993] Gonzalez, A. J. and R. H. Ahlers, "A Context-based Representation of Tactical Knowledge for Use in Simulation-based Autonomous Intelligent Platforms", Proceedings of the 15th Annual Interservice/Industry Training Systems and Education Conference, Orlando, FL, November, 1993, pp. 543-552.

[Gonzalez, 1994] Gonzalez, A. J. and R. H. Ahlers, "A Novel Paradigm for Representing Tactical Knowledge in Intelligent Simulated Opponents" Proceedings of the 7th International Conference in Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Austin, TX, May 1994, pp. 515-523.

[Klee, 1991] Klee, H. I., "Development of a Low Cost Driving Simulator", *Technical Report*, Department of Computer Engineering, University of Central Florida, 1991.

[Schank, 1977] Schank, R. C., and Abelson, R. P., 1977, *Scripts, Plans, Goals and Understanding*, Erlbaum, Hillsdale, NJ, 1977.

[Thorndike, 1984] Thorndike, P. W., and Wescourt, K. T., "Modeling Time-stressed Situation Assessment and Planning for Intelligent Opponent Simulation," *Final Technical Report* PPAFTR-1124-84-1, sponsored by the Office of Naval research, July, 1984.

6. Author's Biographies

Avelino J. Gonzalez is an Associate Professor at the Electrical and Computer Engineering Department at the University of Central Florida. His area of research is in intelligent simulations, specifically, automated intelligent platforms in a training simulation. He has a PhD in Electrical Engineering from the University of Pittsburgh, and a Bachelor's and Master's degrees also in Electrical Engineering from the University of Miami.

Robert H. Ahlers is a Research Psychologist with the Human/Systems Integration Division of the Naval Air Warfare Center, Training Systems Division. He has managed research projects concerned with the application of knowledge-based modeling to the simulation of intelligent agents within a training environment. He graduated from the University of Virginia with B.A. and M.A. degrees in experimental psychology and from North Carolina State with a Ph.D. in Human Factors.

Automated Agents that Learn and Explain Their Own Actions: A progress report

[1]Sakir Kocabas, [2]Ercan Oztemel, Mahmut Uldudag, and Nazim Koc
Department of Artificial Intelligence
Marmara Research Center, PK 21, Gebze, 41470 Turkey
[1]uckoca@cc.itu.edu.tr
[2]eomam1@yunus. mam.tubitak.gov.tr

1. Abstract

Computer generated agents need to be able to learn meaningful actions in various tactical situations and explain the reasons behind such actions. Different inductive methods have been tried by a few research groups in teaching actions to such agents in tactical air simulations. There have also been some attempts to enable the intelligent agents explain reasons behind their own actions in the form of debriefing records. However, previous research has left the integration of learning and real time explanation as an open issue. The use of inductive methods in teaching tactically meaningful actions makes it rather difficult to integrate learning and explanation. In our research, we use explanation-based generalization in teaching meaningful actions and their real time explanations to an intelligent target. Our research aims integrating artificial intelligence in 1-v-1 air combat scenario as part of an international EUCLID project for building a distributed intelligent simulation system.

2. Introduction

Recent research on computer generated agents focus on using artificial intelligence (AI) techniques in controlling such agents. Several research groups have studied the application of AI techniques in various aspects of air to air combat. These efforts include the application of neural networks for acquiring air combat decision-making skills (Crowe, 1990); automated agents for beyond visual range (BVR) tactical air simulation (Rosenbloom et al., 1994); knowledge based decision aiding for BVR combat with multiple targets (Halski et al., 1991); generating agent goals in an interactive environment (Jones et al., 1994); and agents that explain their own actions (Johnson, 1994).

A large part of the current research relies on static knowledge based methods rather than machine learning techniques which enable the dynamic acquisition of the knowledge and skills of human behavior in tactical situations such as in air combat.

In our research we attempt to implement explanation-based learning (EBL), a deductive machine learning technique, in teaching computer generated agents to perform intelligent behavior in BVR and close combat. This study is carried out as part of a joint EUCLID project RTP 11.3 which aims building a distributed simulation system capable of integrating C3I functions and AI techniques. The project uses ITEMS as the simulation environment.

Explanation-based learning has been one of the extensively investigated machine learning methods in artificial intelligence (see, e.g., Mitchell et al., 1986). Different versions of EBL has been applied to a variety of tasks, such as learning concepts, control rules, planning and scheduling, but the majority of these applications are in small domains.

3. The Task Domain

The aim of our research is to develop techniques to create AI targets (AIT) capable of performing intelligent behavior in tactical air combat. The tactical behavior includes BVR and close combat, in a BARCAP (Barrier Combat Air Patrol) scenario for an F16 plane. The task is the intelligent control of the AIT from an AI station connected to the main simulation system via Ethernet (see, Figure 1.)

The ITEMS simulation system is capable of representing a large number of independent agents called "scenario elements" or "targets" in a real-time 3-D environment including geographical, atmospheric and terrain data. In a scenario, the scenario elements can be controlled by

[1]Also affiliated with: Department of Space Sciences and Technology, ITU, Maslak, 80626 Istanbul, Turkey
[2]Also affiliated with: Department of Industrial Engineering, SAU, Esentepe Kampusu, Adapazari, Turkey

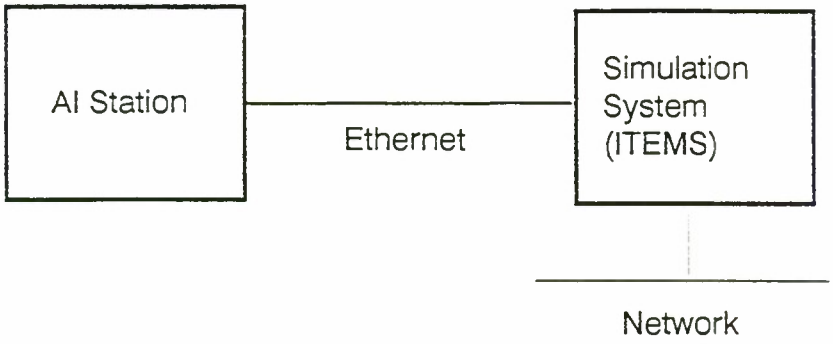


Figure 1. The hardware structure for the intelligent control of scenario elements.

human operators or control programs. The ITEMS system itself has rule based facilities for developing control programs for creating automated agents.

The acquisition of knowledge and skills for complex real-time behavior as in tactical air combat is a difficult task. Acquisition and handcoding of rules for such behavior is rather tedious, as it is difficult to foresee all possible interactions. Therefore, machine learning methods need to be used for the acquisition of such knowledge and skills. Some inductive methods have been used in acquiring the rules of intelligent behavior, e.g. from flight data obtained from exercises (see, e.g., Crowe, 1990; Sammut et al., 1992). However, inductive methods require a large number of training examples in order to support reasonably acceptable behavior.

Additionally, it is difficult, by inductive methods, to integrate capabilities for the intelligent agent to explain its own behavior in every tactical situation. Behavioral explanations for intelligent agents have been studied by Johnson (1994) using SOAR, but the explanations provided by Johnson's Debrief system are post-flight explanations, rather than real time explanations.

We have been developing an integrated system called RSIM, for controlling an F16 plane in the ITEMS simulation environment in an intelligent and human-like way. A prototype of RSIM has been tested on a 2-dimensional simulation medium for BARCAP mission in 1-v-1 tactical situations. The RSIM prototype is capable of learning tactical behavior at training sessions, and producing and explaining its agent's behavior in real time during the execution of a mission.

4. Control Structure

In order to explain RSIM's operation, we will describe the program in terms of its problem space, its subsystems, and its inputs and outputs. As shown in Figure 2, the program consists of three subsystems: Situation-Assessment, Action-Management, and Learning operators.

The simulation system which is controlled by RSIM, is a program which creates and moves simple objects (or targets) in a 2-d space in accordance with the inputs received from RSIM. The inputs indicate the positions and headings of the targets and the missile fires. The simulation system calculates the positions of targets by their intended headings, and moves targets to those positions. It receives inputs in cycles, and operates continuously.

RSIM's Situation-Assessment operator enables the system to continuously assess tactical situations in the simulation environment. The problem space of RSIM consists of two targets, an AI target (AIT) and a man controlled target (MCT) moving in a 2-d space. Both targets have the same degree of freedom of movement. Each target can move at the constant speed of one pixel at 1/2 second. There are 12 state variables for the targets. The variables and their types are as follows:

x,y, coordinates (AIT/MCT)	(integer)
Headings (AIT/MCT)	(8 directions)
Range (AIT-MCT)	(real)
Positional angle (AIT-MCT)	(real)
Time	(real-time)
Missile range	(integer)
Missile fired	(integer)

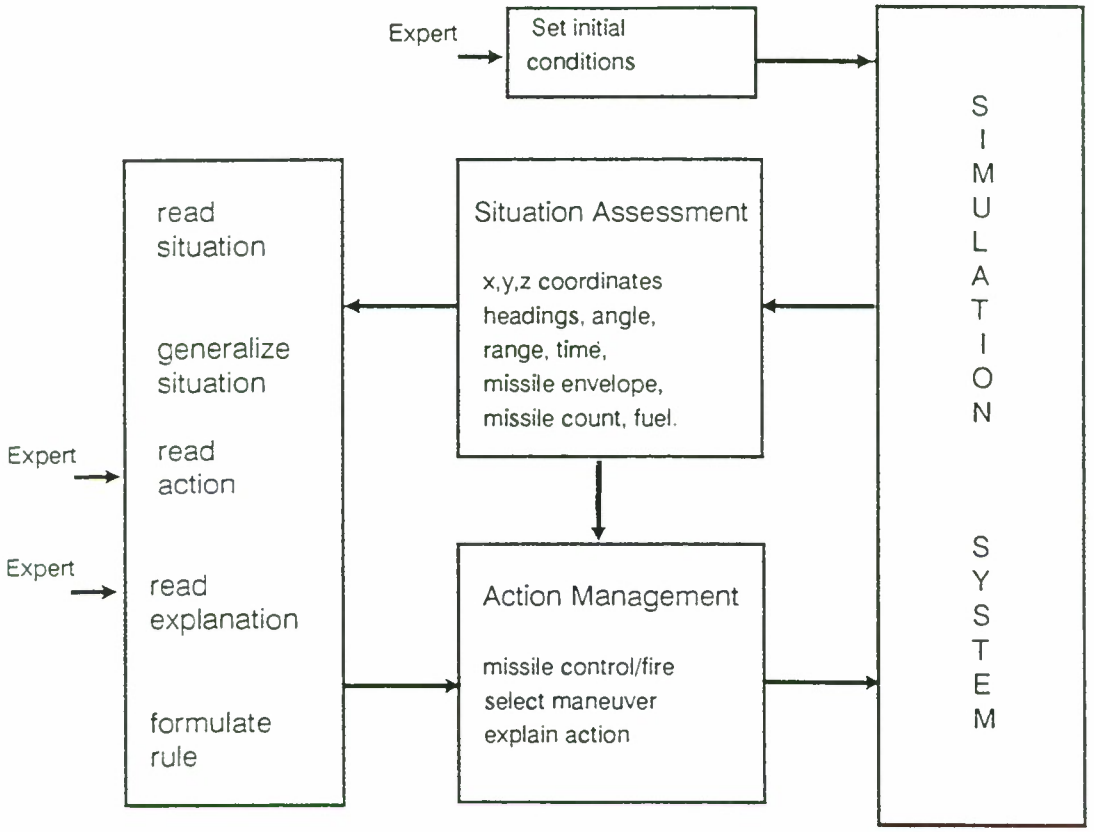


Figure 2. Control structure of RSIM.

The values of the state variables at each instant, determines the problem situation. As the targets change their positions every 1/2 second, the problem situation changes accordingly. At every cycle, RSIM has to make situation assessment, and has to decide which action to take. The Situation-Assessment operator reads the coordinates of the targets and calculates the distance and the positional angle between the targets. The state variables and their values are sent to a message list by the Situation-Assessment operator. This message list is read by the Action-Management operator.

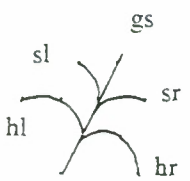


Figure 3. Five simple maneuevers for RSIM agents

The Action-Management operator has three functions: Select-Maneuver, Missile-Control, and Explain-Behavior. The Select-Maneuver function decides the action to be taken by the AIT, by reading the message list and matching the operational variables in the message list with the action rule set. The rule that matches is selected as the action rule.

Each action rule points to a simple maneuver, where each maneuver consists of a four pixel motion. As shown in Figure 3, there are five such simple maneuvers: go straight (gs), soft turn right (sr), hard turn right (hr), soft turn left (sl), and hard turn left (hl). Each maneuver lasts two seconds.

All messages of the Action-Management operator, including the explanations, are sent to the simulation system. The maneuver messages are applied by the simulation system in single-step actions. For example, a command message that says "apply gs maneuver" (go straight), is performed by moving the target four pixels, one pixel at a time, maintaining the target heading.

Although the selected maneuvers last two seconds, situation assessments continue to be made at every cycle of 1/2 seconds. In this way, when AIT enters in a

missile firing position during a simple maneuver, the Missile-Control function fires a missile if the latter is available.

Conditions: Distance is D6,
Angle is A5,
Heading (AIT) is E.

Action: Apply maneuver GS.

Explanation: Target detected. Approach target.

Figure 4. Example of a rule generated by RSIM.

The Action-Management operator can explain the reasons for selecting a particular maneuver by sending a message to the simulator to be displayed in a screen window during the execution of that maneuver. In this way, the behavior of AIT is explained for every simple maneuver in a series of maneuvers.

RSIM has a learning subsystems which learns action rules for the AIT by an explanation-based generalization (EBG) mechanism. This method relies on deductive inference based on the following: i) a goal concept, ii) a domain theory, iii) training examples, and iv) a description of the form in which the learned concept is to be expressed, i.e., the operationality criterion. Unlike inductive methods, EBG constrains the search by relying on knowledge of the task domain and of the concept under study. After analyzing a single training example, this method is able to generate a valid generalization of the example along with a deductive justification of the generalization in terms of this domain knowledge.

RSIM's EBG operator generalizes problem states into general problem states which are predefined by using domain knowledge. In a training session, the program finds the generalization of each problem state, and associates the action taken by the instructor with the generalized problem state. Action rules and explanations are learned during training sessions in an incremental fashion.

Action rules are if-then rules that match situations with simple maneuvers. At each problem state, operational variables in the message list which is periodically updated by the Situation-Assessment operator, define the current situation. If no rule exists to match the current situation, then the Learning operator asks the instructor which maneuver to select. The Learning operator then generalizes the current situation, and records it as the conjunctive conditional part of the rule whose action part is the selected maneuver. The generalization consists of generalizing the values of the operational situation variables form real values to a predetermined range. In this way, e.g., the distance and angle between the two targets are mapped into particular ranges of distance and angle.
The instructor also gives an explanation as to why that particular maneuver was selected. This explanation is associated with the action rule generated for the current situation as the reason for the selection that rule. An example rule is shown in Figure 4. The rule in this figure says that when the distance between the targets is within D6, the positional angle is within A5, and the heading of the AIT is east, then continue to go straight. The reason for this particular maneuver is that the target has been detected and the intention is to approach the target.

RSIM can apply the rules that it has generated as soon as the matching situations arise. In other words, the program generates and uses its rules in a dynamic way. Once the training session ends, learned rules can be stored in a rule file for future use. Figure 5 shows the behavior of RSIM against an automated target when the system had 85 rules in its rule base.

5. Discussion

RSIM's methods of learning are similar to that of LEX (Mitchell et al., 1986) in that it learns to associate problem states with operations or actions. However, unlike LEX which has been applied to static problems such as learning to solve linear equations and integrals, RSIM operates in a dynamic environment. Additionally, the number of RSIM's problem states (over 400) and associated action rules are much larger than that of LEX (about 25).

On the other hand, RSIM controls objects in a 2-d space, and needs to be further developed for objects moving in a 3-d space. The system is being tested on the Flight Simulator for 1-v-1 air combat, where the simple maneuvers are redefined, and explanations are associated with the rules after training sessions. Beside EBL techniques, algorithmic techniques are also being tested for BVR and close combat maneuvers. This seemed necessary for comparison for deciding where each technique is more efficient.

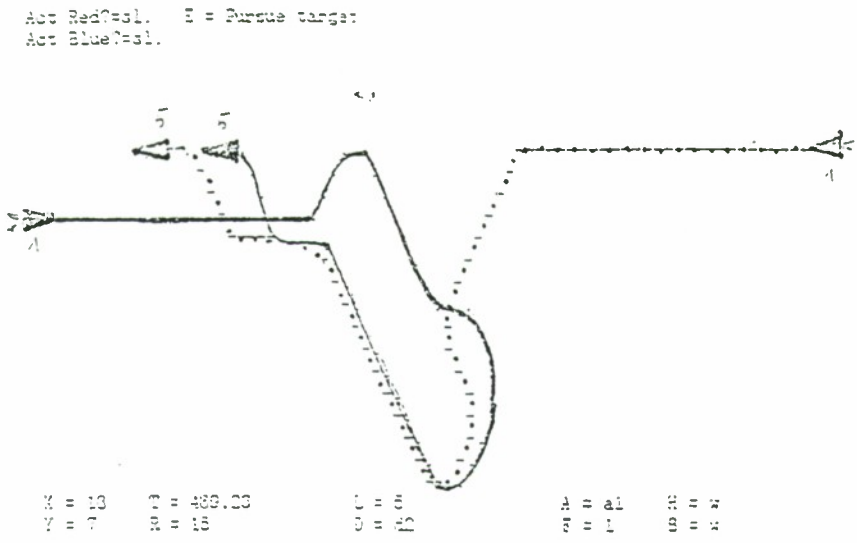


Figure 5. The behavior of the AIT (dark triangle) controlled by RSIM, against an automated target.

As has been described, RSIM can explain its behavior during a scenario in a continuous way. Explanation of agent behavior in flight simulation has been the subject of a recent paper by Johnson (1994), but his explanations are post-flight explanations rather than real-time. Sammut et al. (1992) have used inductive methods for generating rules to control a fixed-wing aircraft in the Flight Simulator, but explanation-based methods have not been applied for such tasks.

Crowe (1990) describes the use of neural nets for learning air combat rules in a prototype. The main advantage of EBL over inductive methods is its ability to use domain knowledge effectively in generating rules of action, and hence its reliance on a far fewer training examples than the latter.

6. Conclusion

In this paper we described a simple real-time intelligent system RSIM which learns how to control a computer generated agent against another one moving in a 2-d space. The system learns its control rules by explanation-based generalization. The learning consists of turning human performed actions in a particular situation into rules of action applicable in a generalized situation. In this way, the program generates a set of condition-action rules which enable an agent to perform intelligent behavior, and explain its own behavior in real-time in a 2-d space.

RSIM has been developed as a prototype for testing the applicability of deductive machine learning methods in 3-d simulation environments.

7. Acknowledgment

This research has been supported by MRC and MOD-R&D under the joint EUCLID project RTP 11.3 led by CAE Electronics GmbH.

8. References

Crowe, M.X. (1990). "The application of artificial neural systems to the training of air combat decision-making skills", In Proceedings of the 12th ITSC., pp. 302-312.

Halski, D.J., Landy, J.R. & Kocher, J.A. (1991). "Integrated control and avionics for air superiority: A knowledge-based decision-aiding system", AGARD CP-424, Madrid 1991, pp 53-1 to 53-10.

Johnson, W.L. (1994). "Agents that explain their own actions", In Proceedings of the 4th Conference on Computer Generated Forces, May 1994, Orlando, Florida.

Jones, R.M., Laird, J.E., Tambe, M. & Rosenbloom, P.S. (1994). "Generating goals in response to interacting goals", In Proceedings of the 4th conference on Computer Generated Forces and Behavioral Representation.

Mitchell, T., Keller, R.M., and Kedar-Cabelli, S.T. (1986). "Explanation-based generalization: A unifying view", Machine Learning 1 (1) 47-80.

Rosenbloom, P.S., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Lehman, J.F., Rubinoff, R., Schwamb, K.B. & Tambe, M. (1994). "Intelligent automated agents for tactical air simulation: A progress report", In Proceedings of the 4th conference on Computer Generated Forces and Behavioral Representation. pp. 69-78.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. (1992). "Learning to fly", Machine Learning Workshop Proceedings, pp. 385-393, Morgan Kaufman.

9. Author's Biographies

Sakir Kocabas is the head of the AI Department at MRC and the project manager for EUCLID RTP 11.3 WP2. Dr. Kocabas has a PhD degree in Information Engineering. His research interests are in the areas of Machine Learning and Discovery.

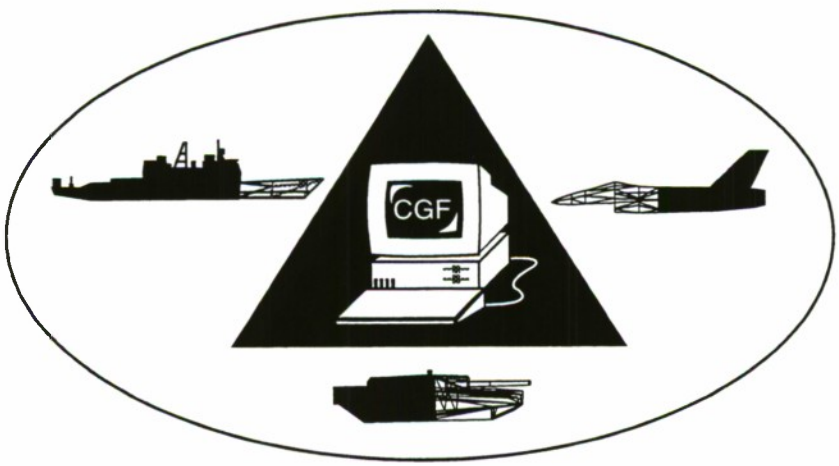
Ercan Oztemel is a researcher at the AI Department of MRC. Dr. Oztemel has a PhD degree in Artificial Intelligence. His research interests are in the areas of Real-Time Expert Systems and Neural Networks.

Mahmut Uludag is a researcher at the AI Department of MRC. Mr. Uludag has a Masters of Science degree in Mechanical Engineering. His research interests are in the area of Real-Time Control of Computer Generated Forces.

Nazim Koc is a researcher at the AI Department of MRC. Mr. Koc has a Masters of Science degree in Symbolic Computation. His research interests are Symbolic Computation, Logic Programming and Machine Learning.

Session 3a: Constructive + Virutal Simulation

Calder, SAIC
Kraus, UCF/IST
Petty, UCF/IST


CGF

Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSD CLCGF

Robert B. Calder, Jeffrey C. Peacock, Jr.
SAIC
486 Totten Pond Road
Waltham, MA 02154
rcalder@bos.saic.com, jpeacock@bos.saic.com

James Panagos
TASC
55 Walkers Brook Drive
Reading, MA 01867
jpanagos@world.std.com

Thomas E. Johnson
Raytheon Company
50 Apple Hill Drive
Tewksbury, MA 01876
tej@swl.msd.ray.com

1. Abstract

One of the major challenges which confronts DIS researchers today is that of integrating simulation systems which are rooted in vastly different domains. These simulations are designed with different architectures and developed for varied goals. Constructive models are developed for performing analytical combat analysis, typically in a standalone environment. Entity-level, DIS simulations are developed for training and testing in a distributed, networked environment. High-fidelity engineering models are developed for engineering analysis of new or prototype vehicle, weapon, or sensor systems in a standalone environment, or as embedded software. In addition, fielded military systems have their own unique origins, approaches, and goals.

In order to successfully integrate systems such as these with varied origins, architectures, interfaces, and goals, many problems need to be solved. While many projects have attempted to integrate some of the above types of systems, few, if any, can claim successful integration and interoperation of all four. The JPSD CLCGF project has successfully demonstrated this achievement.

In this paper, we present work which has been performed on the JPSD CLCGF project with an emphasis on the integration between constructive, virtual, live, and engineering simulations.

2. Introduction

2.1 The JPSD Program

The Joint Precision Strike Demonstration (JPSD) program's goal is to introduce and implement new technologies into the defense arena that can address and correct precision strike deficiencies. To facilitate this goal, the JPSD program has created a simulation environment which is used to evaluate technologies, train users, and perform experiments necessary to reduce sensor-to-shooter timelines and to attack high-value, time-sensitive targets. As part of this environment, the JPSD program has sponsored the construction of the Corps Level Computer Generated Forces (CLCGF) system.

The primary purpose of the CLCGF is to provide the corps level simulation environment for DIS exercises in which the above mentioned program goals can be carried out. The CLCGF is used during the JPSD exercises to simulate maneuver and artillery units contained in an Army corps. The simulated units provide stimulus for and interact with tactical hardware systems and their operators, and interact with high-fidelity, engineering simulations. The CLCGF has been created by integrating the Eagle constructive simulation with the ModSAF entity-level simulation.

2.2 The CLCGF System

Entity-level simulations represent each entity which exists on the virtual battlefield at the individual platform level. They typically represent entities from the individual platform level up to the company level. They use the DIS protocol to interact with other entity-level simulations, and simulate the physical

characteristics of each entity to determine battlefield outcomes. On the other hand, constructive simulations represent groups of entities as single, aggregate unit objects. They typically represent units at the company or battalion level up to the division or corps level. They are typically not designed to interact with other simulations, but instead simulate the entire battlefield internally, and use Monte-Carlo techniques to determine battlefield results.

The DIS environment has traditionally included only entity-level simulations. It has provided a sound environment for small-scale, tactical troop training, as well as a potential testbed for evaluating new vehicle and weapon systems. However, simulating the effects of entity-level simulations in corps level operations has remained beyond the reach of the DIS environment, due to network bandwidth and computer resource constraints. Using current network and computer technology, a traditional DIS exercise is simply not capable of supporting a corps level operation. This was the primary motivation for creating a CLCGF which utilizes both constructive and entity-level simulations. Transmission of unit state data at the aggregate level is a key factor which decreases network load by significantly decreasing the number of PDUs transmitted in a large-scale exercise. If DIS is to support a 100,000 entity exercise, representation of some units on the battlefield as aggregates is likely.

The simulation engine of the CLCGF has been built by integrating the constructive, aggregate-level simulation Eagle, with the virtual, entity-level simulation ModSAF. This simulation engine interacts with various live, tactical hardware systems, including: the All Source Analysis System (ASAS) Warrior and Ground Station Simulator (GSS) for presentation of the tactical battlefield situation to the operator and potential target nominations; and the Automated Deep Operations Coordination System (ADOCS) for the creation and assignment of fire missions. The simulation engine interacts with the STRIKE engineering-level simulation, which simulates the deployment and flyout of smart submunitions. It also interacts with the TAFSM simulation, which it utilizes to simulate the deployment and flyout of various smart submunitions. In addition, the CLCGF interacts with various other DIS simulations, such as the Warbreaker SimCore simulation.

In order to allow military training and analysis of scenarios of interest to JPSD, the CLCGF must generate a full corps-level exercise. To accomplish this goal, many technical challenges need to be addressed. These involve issues such as efficient incorporation of aggregate units into DIS, effective incorporation of DIS entity-level information into constructive simulations, development of a dynamic aggregation/deaggregation protocol, interaction between constructive and entity-level simulations, and interaction between a constructive/virtual simulation, live systems, and engineering-level simulations. The work performed on the CLCGF to date has focused on these fundamental goals.

2.3 CLCGF Interaction with other JPSD Systems

In order to create a test and evaluation environment in which to conduct JPSD experiments and studies, the requisite constructive, virtual, and engineering-level simulations must interoperate with one another, as well as with current and future fielded, tactical systems used in Army Corps operations. A block diagram of the CLCGF system and the non-DIS systems with which it interfaces is shown in Figure 1.

The CLCGF system consists of the linkage between Eagle, the SIU (Simulation Integration Unit - whose primary function is to link Eagle to the DIS network), and ModSAF. It is responsible for simulating the entities and aggregate units on the corps battlefield, presenting a plan view display of the map and all units and entities, interacting with other DIS simulations, and interfacing with the other non-DIS systems shown in the block diagram. The ASAS Warrior and GSS systems are used to present a picture of the tactical battlefield situation to an operator (via communication feed from a JSTARS radar), and to initiate precision strike target nominations. The ADOCS system is used to create, monitor, and assign fire missions to corps artillery assets. The STRIKE simulation is used to simulate the deployment, flyout, and impact of smart sub-munitions. These systems and their interactions will be described in the remainder of this paper.

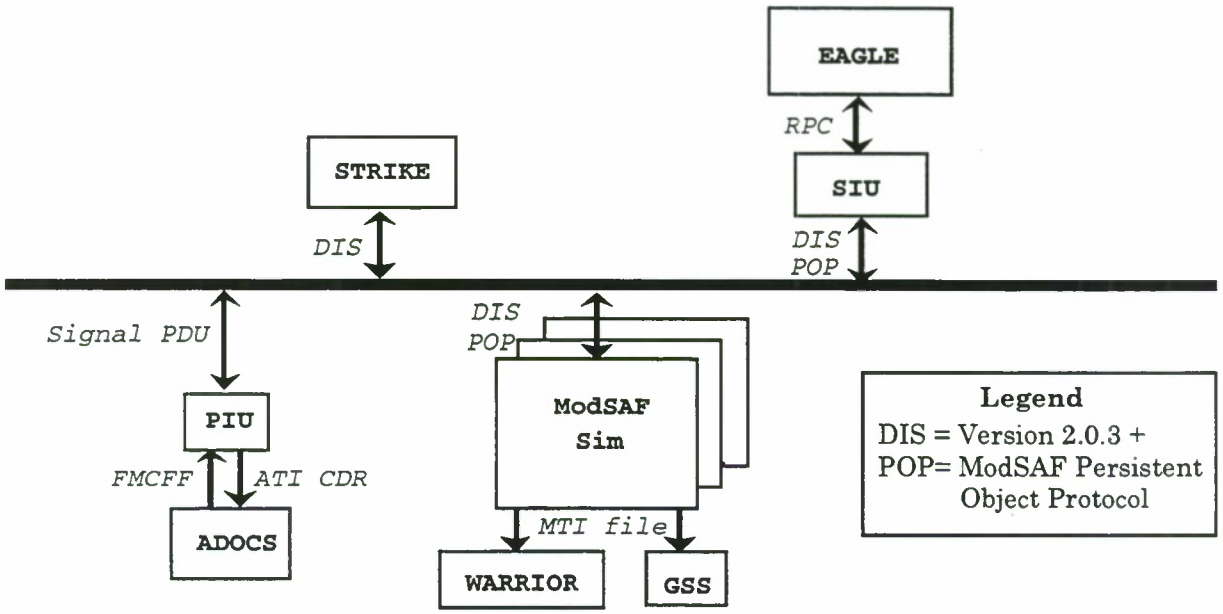


Figure 1: CLCGF Interface Block Diagram

3. Integration of Constructive, Virtual, Live, and Engineering Simulations

Many issues need to be addressed in integrating simulation systems with different origins, since they were designed and developed with different goals in mind. Some, such as constructive models, were developed for performing analytical combat analysis in a standalone environment. Others, such as DIS entity-level, virtual simulations, were developed for training is a distributed network environment. Still others, such as high-fidelity engineering models, were developed for engineering analysis of new or prototype vehicle, weapon, or sensor systems in a standalone environment, or as embedded software. The technical approach and level of fidelity of the resultant simulations varies greatly. In addition, fielded military systems have their own unique origins, approaches, and goals.

In order to integrate these systems of varied origins, architectures, interfaces, and goals, problems such as communication, time synchronization, level of fidelity, and terrain and environment correlation need to be solved.

Communication: Constructive models and engineering-level simulations are typically developed as standalone systems. DIS simulations are developed with the capability to communicate with one another via the DIS protocol. Tactical military systems are typically capable of communicating with other relevant tactical systems via some military standard protocol (e.g. TADIL, TACFIRE, etc.).

Time Synchronization: Constructive models typically run faster-than real time in order to simulate outcomes of events which occur over the course of a large-scale battle in a relatively short period of time. Some constructive simulations are capable of running in real time. DIS simulations must run in real time, since DIS exercises typically include humans in the loop. Some DIS simulations are capable of running faster-than real time. High-fidelity, engineering-level simulations typically run slower-than real time, since they are modeling in software some processes and functionality which will be implemented in hardware in a real system, and since they need to allow for inspection into the system. Tactical military systems run at real time since they are operating in the real world and require a human operator.

Level of Fidelity: Constructive models represent battlefield entities as aggregate units, usually at or above the company level. DIS simulations represent battlefield entities at the platform level, and the level of fidelity is typically low, but does vary from simulation to simulation. Some computer generated forces systems are capable of performing coordinated entity behaviors up to the company level, and varying the level of fidelity of the platforms which they simulate. Engineering-level simulations are high-fidelity models of an actual or prototype platform

system. Tactical military systems are high-fidelity systems which are built to the specifications which are needed for a given real world application.

Terrain and Environment Correlation: Constructive models typically represent the terrain as large homogeneous areas of mobility and intervisibility characteristics, with aggregated features. DIS simulations represent the terrain at a higher level of fidelity, with sampled elevation data and small obstacle and feature information. Engineering-level simulations typically represent the environment in which they execute at a very high level of fidelity, since their purpose is to perform analysis of how a platform will perform in the real world. Tactical military systems operate in the real world, and therefore do not require any model of the environment.

In addition, there are many other areas in these systems in which a common concept or feature is represented. But even these aspects, which seem to be non-issues on the surface, require some translation from one system to the other, since the various systems typically implement their solutions using different approaches.

All of these issues serve to make the challenge of integrating constructive, virtual, live, and engineering systems a difficult one.

4. CLCGF Simulation Engine

4.1 Constructive/Virtual Simulation Linkage

Some of the issues involved in integrating constructive and virtual systems have been documented and addressed in implementations by DIS researchers (Calder et. al. 1994, Karr et. al. 1994, Karr et. al. 1993, Hardy et. al. 1993). Various approaches have been taken to solve these problems, and some achievements have been made. All successful constructive/virtual linkage projects have:

- represented aggregate units in DIS by transmitting aggregate unit state information on the DIS network

- represented virtual entities in the constructive model by forwarding entity state information from the DIS network to the constructive model

- deaggregated constructive model aggregate units into virtual simulation DIS entities

- re-aggregated virtual simulation DIS entities into constructive model aggregate units

- communicated aggregate unit orders to DIS entities upon deaggregation

- reported deaggregated unit status to the constructive model

- provided indirect fire interaction between aggregate units in the constructive model and DIS entities in the virtual simulation.

The implementation of these solutions has varied from point solutions to robust architectures.

However, there still exists a second category of issues and problems which have not been solved, or even seriously addressed, to date. These include improvements in terrain correlation between constructive and virtual terrain databases, implementation of direct fire between virtual entities and constructive aggregate units, incorporation of constructive units as aggregate entities in the virtual simulation, time synchronization between constructive and virtual models, control of spreading deaggregation, and, in general, the elimination of combat results correlation error.

The CLCGF project has implemented robust solutions to the first category of problems. We have also begun to address some of the problems in the second category, most notably in the incorporation of constructive units as entities in the virtual simulation and in the control of spreading deaggregation. These solutions will be discussed later in this paper.

4.2 Eagle/ModSAF Simulation Linkage

The simulation engine of the CLCGF has been built by integrating the constructive, aggregate-level simulation Eagle, with the virtual, entity-level simulation ModSAF. Use of these existing systems enabled the JPSD program to create a useful CLCGF quickly and with less risk. These two simulations are a good match since Eagle can simulate aggregate units down to the company level, while ModSAF can simulate entities in units up to the company level. In addition, work on integrating Eagle with another CGF system, the IST SAF, had already made a great deal of progress and provided a solid interface between Eagle and the virtual world. In addition, ModSAF is a fully distributed system and has a mechanism, the Persistent Object (PO) protocol, for implementing distributed command and control of CGF and for representing information about "persistent" objects (e.g. missions, graphics, overlays, etc.).

In order to link Eagle with ModSAF, a mechanism was needed to allow the two simulations to communicate. Previous work in the Integrated

Eagle/BDS-D project linked Eagle with the IST SAF (Karr et. al. 1994/1993). This project created an interface to Eagle which facilitated transmission and receipt of aggregate unit and deaggregated unit state information to and from Eagle, as well as other relevant information. This interface was called the Simulation Integration Unit (SIU). We decided to reuse the SIU interface mechanisms and message formats established by this project, in order to get maximum reuse of their efforts and to establish a working system as quickly as possible. In addition, reuse of this software allowed us to make significant progress without the need for changes to the Eagle software.

Given the SIU interface mechanism along with the existing interface message formats, we needed to provide the linkage to ModSAF. The easiest way to do this was to use ModSAF itself as the cornerstone, since many of the features and functionality needed for an Eagle/ModSAF SIU were already present in ModSAF (e.g. DIS interface, PO protocol, GUI, etc.). Given this decision, we incorporated the Integrated Eagle/BDS-D SIU interface mechanisms and message formats into ModSAF. The resultant SIU for our project is a modified version of ModSAF.

The SIU performs many functions including communication between Eagle and DIS, communication between Eagle and ModSAF, simulation of aggregate unit entities, approximation of aggregate unit position between Eagle time steps, display of aggregate units on the ModSAF PVD, dynamic aggregation and deaggregation of units via various mechanisms, and intelligent vehicle placement in the deaggregation process.

Figure 2 shows the interaction of the components which comprise the CLCGF simulation engine. Eagle communicates via remote procedure calls (RPCs) with a process called the Eagle RPC Server. The information passed over this interface is described in section 4.3.1 below. The Eagle RPC server runs on the same host as the SIU, and communicates with the SIU via shared memory. The information passed over this interface is described in section 4.3.1 below. The SIU communicates with one or more ModSAF simulators via the Persistent Object (PO) protocol, and with other DIS simulations (including ModSAF) via the DIS protocol.

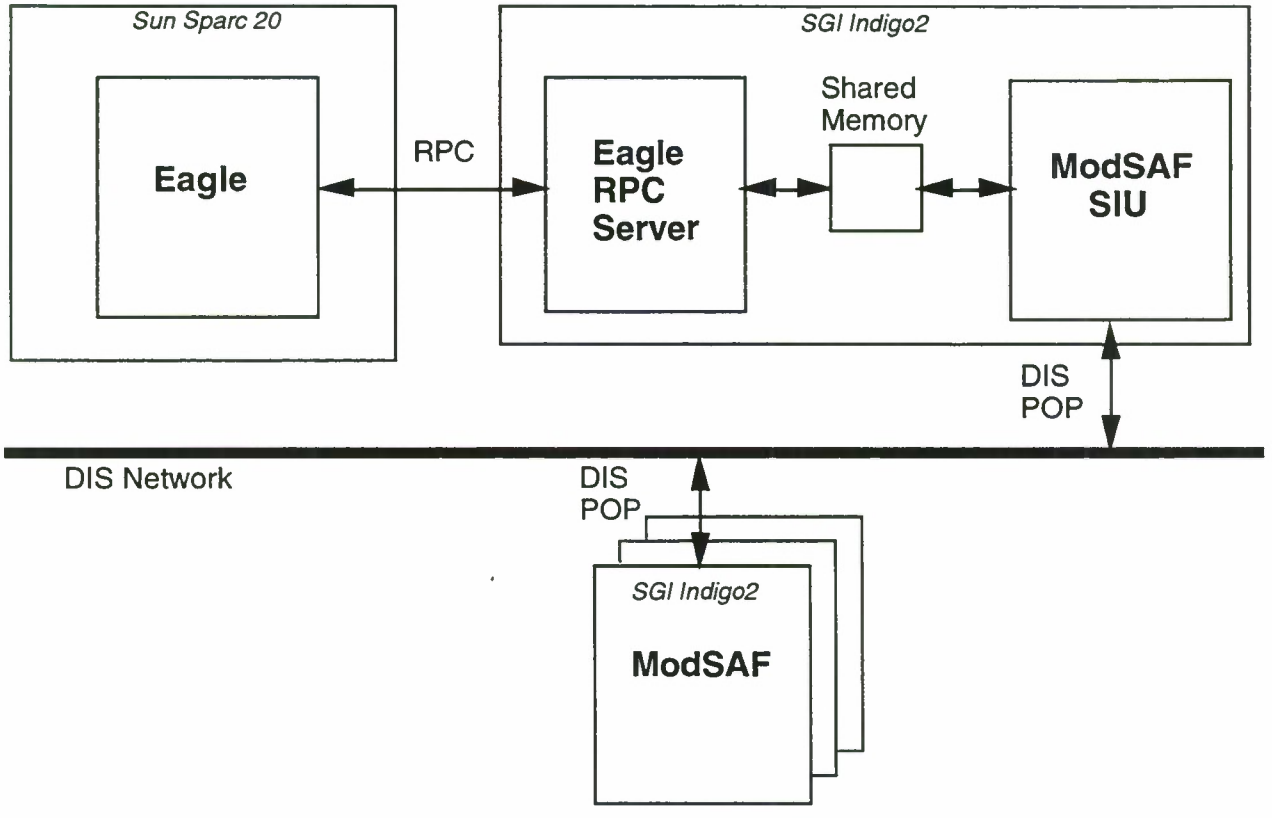


Figure 2: CLCGF Simulation Engine Components and Interfaces

In order to integrate Eagle and ModSAF into the CLCGF for JPSD, we needed to modify the application software of both simulations, and create a new, ModSAF-based SIU. The design and implementation of the resultant features are discussed in detail below.

4.3 CLCGF Features

4.3.1 Eagle<->SIU Interface

For Eagle to be capable of limited participation in a DIS exercise in the CLCGF, the SIU needs to serve as the link between Eagle and the DIS network, as well as the link between Eagle and ModSAF. This was accomplished by implementing an Eagle<->SIU communication interface in the ModSAF-based SIU. This required creating a new ModSAF library, "libsiu", which implements this interface. It sends and receives messages and data to the Eagle RPC Server via shared memory. The major communication functionality supported by "libsiu" includes:

- Creation of Eagle aggregate units in the virtual world, to facilitate transmission of aggregate unit state information on the DIS network.

- Update of Eagle aggregate unit state information, to facilitate transmission of accurate aggregate unit state information on the DIS network.

- Reporting of deaggregated unit state information to Eagle to provide a mechanism for status information, such as that typically contained in situation and spot reports, to be transmitted to Eagle to keep Eagle informed of the state of deaggregated units in the virtual world.

- Reporting of aggregation/deaggregation status to Eagle for all units, to keep Eagle informed of which units are under its control.

- Aggregation/deaggregation of a unit upon Eagle request, to allow Eagle to initiate aggregation and deaggregation of units based upon certain criteria.

- Deployment of indirect fire from Eagle's constructive world into the DIS virtual world, in order to effect DIS entities. The SIU transmits fire and detonation PDUs on behalf of Eagle for all Eagle fire missions. This allows constructive indirect fire to effect the virtual world.

- Requesting that fire missions received from the virtual world (via simulators or live, tactical hardware) be passed to Eagle and performed as Eagle fire missions.

4.3.2 Aggregate Unit Protocol

A protocol for aggregate units and their aggregation/deaggregation has been defined and implemented. It specifies the format and the transmission, receipt, and processing requirements of DIS 2.0.3 experimental PDUs to facilitate aggregate units. Specifically, two new PDUs have been defined: the Aggregate State PDU (ASPDU) and the Deaggregation Request PDU (DRPDU).

The ASPDU is similar in usage and purpose to the DIS Entity State PDU, but is used only for aggregate units. It allows Eagle aggregate units to be broadcast on the DIS network similar to the way entities are normally broadcast in traditional DIS exercises. The ASPDU for each aggregate unit is transmitted by the SIU every five seconds and contains the following fields: entity id, unit type, unit marking, aggregate state (i.e. aggregated or deaggregated), position, orientation, velocity, formation, extent, number of entities in the aggregate, and subordinate entity ID's (when the aggregate unit is deaggregated).

Aggregation and deaggregation in the CLCGF is managed by the SIU. Aggregation and deaggregation can be initiated by the constructive simulation (Eagle), any virtual simulation on the DIS network (e.g. ModSAF), or the SIU (via event-driven mechanisms). Deaggregation is initiated by the transmission of a DRPDU. This DRPDU is transmitted periodically to maintain deaggregation of an aggregate unit. Re-aggregation of a deaggregated unit is initiated by ceasing transmission of DRPDUs for that unit.

4.3.3 Aggregate Unit Incorporation in DIS

Aggregate unit state information is broadcast on the DIS network using the aggregate unit protocol defined above. In order to generate the information contained in the ASPDU, however, it is necessary to maintain the Eagle aggregate unit state in the SIU. This is accomplished in the ModSAF SIU by handling aggregate units similar to the way in which ModSAF handles entities.

At scenario start, Eagle sends the initial state of each aggregate unit to the SIU, and the SIU creates a local aggregate simulation unit. This local aggregate unit is comprised of simply an aggregate hull, and is entered into the SIU's vehicle table. It ticks similarly to the way ModSAF entities do, but does not execute any ModSAF tasks, since Eagle controls its behavior.

Eagle runs in real-time at one or two minute time steps in our scenarios, so the SIU receives aggregate unit state updates every one or two minutes. Since it is desirable to broadcast state information more frequently in DIS, the SIU performs a local entity approximation of an aggregate unit's position each time the aggregate unit ticks. Presently, this is simply a linear approximation of position along the last velocity vector reported. In the future, we plan to improve this approximation by considering the routes, phase lines, and orders which the aggregate unit is executing to compute its position between Eagle time steps.

The periodic transmission of ASPDUs by the SIU enables ModSAF, as well as other DIS simulations, to consider aggregate units as remote entities. We have modified ModSAF to receive and process the ASPDUs and display aggregate units on the PVD. This allows the operator to see the entire battlefield of aggregates and entities, and to perform aggregate unit operations (e.g. aggregations and deaggregations) as described below.

4.3.4 Dynamic Aggregation and Deaggregation

This section will provide an overview of the dynamic aggregation and deaggregation capability of CLCGF. For more details, see (Calder et. al. 1995) in these proceedings, which is devoted entirely to this subject.

Aggregation and deaggregation in the CLCGF is dynamic based upon events which occur during the exercise, or upon initiation by an SIU operator or ModSAF operator at a GUI. The system does not rely upon predetermined high-resolution areas or spheres of influence to enable deaggregation, but it does not preclude these mechanisms from being used (Karr et. al. 1993, Hardy et. al. 1993). The mechanisms currently implemented which trigger deaggregation include assignment of a fire mission to an MLRS unit (via live, tactical hardware or the ModSAF GUI), Eagle request, ModSAF operator request, and SIU operator request.

When an Eagle aggregate unit is deaggregated into ModSAF entities, the ModSAF unit is automatically assigned a ModSAF mission based upon its mission in Eagle. Operations orders (OPORDs) are passed from Eagle to the SIU upon deaggregation. However, the SIU does not currently parse the entire OPORD to automatically construct the ModSAF unit mission. Instead, the SIU uses the current operational activity of the Eagle unit, maps it to a ModSAF taskframe, creates the taskframe (using other information from Eagle such as heading, speed, formation, etc.), and assigns the taskframe to the deaggregated ModSAF unit. The ModSAF unit immediately begins execution of the taskframe. In the future, we plan to fully parse the OPORD to automatically construct and assign a consistent mission to the ModSAF unit.

When an Eagle MLRS unit is deaggregated to perform a requested fire mission, execution of a new mission is implied for the deaggregated unit. For example, Eagle MLRS launcher units are deaggregated when the SIU receives a call for fire from a live ADOCS system. In this case, a new mission is implied for the deaggregated unit, which is to perform the requested fire mission. This mission is automatically constructed by the SIU as a ModSAF taskframe, assigned to the deaggregated unit, and executed by the deaggregated unit.

When a deaggregation occurs, formation templates are accessed for the aggregate unit which is to be deaggregated. Once the template has been accessed, an intelligent entity placement algorithm is exercised to modify the positions of the entities to be more realistic. For example, if the unit was performing a roadmarch, then each of the entities is placed on the nearest road (if one exists) in a column formation. Another component of this intelligent entity placement is that vehicles will not be placed on the terrain at locations where they would not be expected to go, such as water, no-go terrain, etc.

Re-aggregation occurs when the event which triggered the deaggregation is completed or upon operator request (if the deaggregation was operator initiated). In any case, it is the responsibility of the simulation which requested the deaggregation to initiate the re-aggregation, since it has the context of why the deaggregation was needed initially.

Our dynamic, event-based aggregation and deaggregation scheme helps control spreading deaggregation, since only those units which intend to interact are deaggregated.

4.3.5 Deaggregated MLRS Unit Capabilities

In order to support JPSD studies, several new entities and models were needed in ModSAF. Much of this work centered around the M270 MLRS vehicle and its associated munitions. ModSAF already supported individual M270 MLRS vehicles, but they did not have the capability to launch ATACMS missiles. This model was therefore modified to be capable of launching ATACMS missiles.

The ATACMS missile flies a ballistic trajectory, and the existing ModSAF missile model did not support ballistic missile dynamics. Therefore, software to simulate a ballistic missile capable of flying in excess of 100 km and releasing submunitions was added to the existing ModSAF missile model. This ballistic model provides the dynamics for the initial

powered boost phase, a roll phase during which the missile rolls to a specified angle to attain the desired trajectory, a ballistic phase where the missile motor has shutdown and the trajectory is computed through apogee, and subsequent re-entry down to either a detonation or impact phase. Throughout the course of the flight, the rotational effects of the earth are considered in computing the trajectory.

The ballistic missile model provides two cases for termination of flight: detonation above the ground, where submunition deployment occurs; and detonation upon impact, where the missile collides with the terrain. In the case of the ATACMS missile, the missile was defined to detonate above the ground in order to simulate submunition dispense at a specified altitude above the target. In this case, the model computes an updated detonation point each tick after the missile has passed apogee, based on the specified target position and the specified impact offset and height of burst. When the position of the missile has been determined to have entered the detonation envelope, a Detonation PDU for the missile entity is transmitted on the network. This Detonation PDU acts as the trigger for a submunition simulation to begin dispense and flyout of submunitions.

In addition to the specific M270 MLRS vehicle modifications, an MLRS battery was created in order to perform deaggregations of Eagle MLRS units, since Eagle only simulates down to the battery level. Individual M270 MLRS vehicles in this unit needed to be capable of performing fire missions independent of one another, as MLRS units do in the real world. However, the ModSAF behaviors for MLRS units did not support this individual firing capability. Therefore, the MLRS unit-level behaviors were modified to allow an M270 MLRS vehicle, which was part of a larger MLRS unit, to fire independently.

5. CLCGF Interaction with other Simulations

The CLCGF simulation engine described above interfaces to other systems in JPSD to provide the simulation environment necessary to fulfill JPSD program goals. The interfaces to these systems are described below, and are shown in Figure 1.

5.1 CLCGF Interface to ASAS WARRIOR and GSS

The ASAS Warrior and Ground Station Simulator (GSS) are both operator manned workstations whose function is to simulate actual ground station modules which monitor the tactical battlefield environment and nominate hostile targets for attack. The ASAS Warrior and GSS both receive tactical situation data from an E-8A fixed-wing aircraft equipped with a Joint Surveillance and Target Attack Radar System (J-STARS). For CLCGF, an E-8A entity was created within ModSAF, based on the known characteristics of the actual E-8A aircraft. Parametric data consistent with that of the E-8A aircraft was used to represent the performance of the E-8A throughout its flight envelope. Flight dynamics for the E-8A model are provided by the fixed-wing dynamics of ModSAF. The E-8A was defined with a radar component to represent the J-STARS radar used for surveillance of the tactical battlefield. The E-8A was also defined with the capability to fly a combat air patrol mission to represent the flight path normally flown during tactical missions.

The J-STARS radar model is a Pulse Doppler (PD) model within ModSAF. Parametric data consistent with the J-STARS PD radar was utilized in defining the model, thus providing a representative detection envelope and operating characteristics. The ModSAF radar model, upon which the J-STARS model is built, provides line-of-sight calculations for all entities in its field of view. Any entity which is determined to be out of range, masked by terrain, or not meeting the other specified parameters of the radar model is not reported as sensed by the J-STARS radar.

In the processing of a sensed aggregate, a pseudo-deaggregation must be performed to decompose the contents of the aggregate into meaningful information for transmission to the ASAS Warrior and/or GSS. Each aggregate is defined by its DIS entity type, number of entities, and formation. From this information and the position of the aggregate, an expansion based on the DIS entity type, which defines the echelon type (e.g. U.S. armor company), and the formation is performed to template the aggregate into a pre-defined pattern, specified within ModSAF, for the number of entities contained in the aggregate. This template, combined with intelligent entity placement algorithms, is then used to compute the location of each pseudo-entity based on the position of the aggregate. In order to ensure correlation between the represented pseudo-deaggregated unit and the same aggregate unit when it is commanded to deaggregate, the same processing is performed for pseudo-deaggregation, with the exception of the creation of the individual entities.

The process of expanding the echelon and formation for each aggregate is time-consuming, and can become a time sink in a large scenario. In ModSAF, the expansion of an echelon type/formation pair always results in the same templating information, i.e. the initial locations relative to the 0,0 point of

the terrain database, facing North. Therefore, by maintaining a copy of the results of the initial expansion for each echelon type/formation pair, it is only necessary to perform the expansion once. Subsequent aggregates which are defined with the same echelon type/formation pair utilize the initial templating information, and compute the correct entity locations based on the position, orientation, and size of the unit.

The CLCGF method of pseudo-deaggregation takes a different approach than that of other constructive/virtual linkage projects. Previous pseudo-deaggregation implementations have transmitted Entity State PDUs on the DIS network when it was desirable for another simulation to obtain entity-level position data for an aggregate unit under control of the constructive simulation (Karr et. al. 1994/1993). In the CLCGF, the responsibility for pseudo-deaggregation is placed on the virtual simulations which need to access the entity-level information for a given aggregate unit. Any simulation which needs entity-level data for an aggregate unit is required to receive and process Aggregate State PDUs, and be capable of decomposing an aggregate into its constituent entities, applying formation templates to place the entities on the terrain, and executing intelligent vehicle placement algorithms to further adjust the positions of the entities. A further requirement exists that all simulations in a given exercise which perform pseudo-deaggregation use the same decomposition data, formation templates, and intelligent placement algorithms to ensure consistent entity-level representations of the same aggregate unit across simulations.

For example, in many CLCGF scenarios there is a JSTARS radar model which is running on a ModSAF E-8A aircraft. The JSTARS is responsible for performing surveillance of the entire battlefield in a corps level exercise. Instead of transmitting entity state PDUs for each entity of each aggregate so that the radar model can run line-of-sight calculations on them, the JSTARS model receives the aggregate state PDUs, and internally pseudo-deaggregates them to the entity level to run the radar calculations. In this way, we do not flood the network with entity state PDUs which are not needed by the majority of the simulations in the exercise.

We feel that this method of pseudo-deaggregation is superior to previously implemented methods. It allows us to keep network bandwidth utilization at a minimum, which is one of the primary issues in implementing a large-scale simulation. It allows only those sensors and systems which need entity state information to compute it, and ensures that they will compute it in a consistent manner (by requiring that they all use the same input data, formation templates, and vehicle placement algorithms). It does not sacrifice any information to achieve these benefits.

Once the list of detected entities and pseudo-entities has been constructed, this information must be passed to the ASAS Warrior and/or GSS. In both cases, a flat file, called an MTI (Moving Target Indicator) file, is created which lists information about the entities detected. While the format of the files is different, the information required by both the ASAS Warrior and GSS is similar. Both systems require a latitude/longitude for each entity and additional information about the entity. For the GSS, the additional information consists of a field indicating whether the entity is Tracked or Wheeled. For the ASAS Warrior, the additional information is the complete DIS entity type, along with the entity's velocity and elevation.

The J-STARS radar model initiates processing of the radar sensed list once every 60 seconds, replicating the update rate of the actual J-STARS radar. Processing of the J-STARS sensed list is spread evenly over the 60 second scan period of the J-STARS, to avoid long ticks during large scenarios where many aggregate units populate the battlefield. The intermediate results of the processing are formatted and written to memory as the radar sensed list is processed incrementally over the scan period. At the end of the 60 second period, the data is written out to the ASAS Warrior and/or GSS interface MTI flat file.

5.2 CLCGF Interface to ADOCS

The Automated Deep Operations Coordination System (ADOCS) uses the TACFIRE message protocol to communicate with other military hardware. In the JPSD program, it is utilized to issue fire missions via the TACFIRE FMCFF (Full Mission Call For Fire) message. Target nominations are sent to the ADOCS from the ASAS Warrior or GSS operator via the TACFIRE ATICDR (Artillery Target Intelligence Coordinate Report) message. The ADOCS operator pairs nominated targets with available artillery assets to compose fire missions for artillery units to execute. All TACFIRE messages used by JPSD are transmitted on the DIS network in the "data" field of the DIS Signal PDU.

The SIU monitors DIS PDUs to detect FMCFF messages. The FMCFF message contains many fields, including identification of the artillery battery for which the fire mission is designated. If a given fire mission is intended for a unit which the CLCGF is simulating, the SIU fully parses and processes the

message. It extracts the target location, shell type, number of rounds, etc. from the FMCFF. The SIU then makes a determination of the area around the target location which can be impacted by the munition being fired. If there are any aggregate units in the area, the SIU begins sending Deaggregation Request PDUs to initiate the deaggregation of these aggregates. In addition, if the unit which was requested to fire is an aggregate, the SIU initiates deaggregation of the unit by sending DRPDUs for it. The SIU then constructs a ModSAF MLRS fire mission for the entity-level MLRS unit, and assigns that fire mission to the MLRS unit. At this point, the ModSAF MLRS unit executes the fire mission.

During initial JPSD demonstrations, the ADOCS TACFIRE database was populated by hand with the ID's of a subset of the US artillery assets that were available for a given scenario. In order to provide a more realistic view of the battlefield for the ADOCS operator, an interface has been added which automatically initializes the ADOCS TACFIRE database with the ID's and locations of all US artillery assets in the scenario. This is accomplished by the SIU sending the TACFIRE AFU (Ammunition Fire Unit) UPDATE message to the ADOCS for all US artillery assets being simulated by the CLCGF in either Eagle or ModSAF. The AFU UPDATE is a standard TACFIRE message used to update the TACFIRE database with fire unit status information.

5.3 CLCGF Interface to STRIKE

The STRIKE simulation is a high-fidelity, engineering-level simulation of a proposed weapon system, which was modified to be capable of running in real-time for use in DIS. It simulates the dispense, flyout, and detonation of Brilliant Anti-Tank (BAT) smart submunitions. The BAT is a submunition which is dispensed from an ATACMS missile, and uses acoustic and infrared sensors to track and attack tanks and other armored vehicles. In order to interact with STRIKE, a protocol for submunition dispense and simulation handoff needed to be defined and implemented in ModSAF. This was accomplished via transmission of four types of PDUs: the Application Action Request PDU, Application Action Response PDU, Entity State PDU, and Detonation PDU.

During ModSAF initialization, the DIS site/application address of the STRIKE simulation computer is initialized via command line input. At initiation of the launch of the ATACMS missile, the missile model transmits an Application Action Request PDU to the STRIKE simulation, passing the entity id of the missile, estimated time of flight, estimated detonation coordinates, and various target specific parameters. This PDU is re-transmitted every 5 seconds for up to 30 seconds, or until receipt of an Application Action Response PDU from the STRIKE simulator with the same site/host address as specified at initialization. Receipt of the Application Action Response PDU notifies ModSAF that the STRIKE simulation received the Application Action Request PDU and is ready to dispense submunitions from the specified ATACMS missile entity.

While the ATACMS missile is in flight, the STRIKE model monitors the ATACMS missile's Entity State PDUs. When it is time for the ATACMS missile to dispense submunitions, it sends a Detonation PDU (as described in Section 4.3.5). Receipt of the Detonation PDU by the STRIKE model triggers it to perform a simulated dispense of the BAT submunitions. The STRIKE model utilizes recent ATACMS missile Entity State PDUs to initialize the BATs with the correct initial attitude and velocity, and begins simulation of the BAT submunitions. The BATs fly to and attack any detected targets in the area.

To account for the potential damage inflicted by the submunition detonations, the ModSAF direct fire damage tables were updated to handle the BAT submunition type. Upon receipt of a BAT Detonation PDU, a table lookup is performed and direct fire damage is calculated for the target entity based on the table.

6. Future Work

Future work will include expanding upon the current Eagle to SIU interface through new interface commands and enhanced functionality. Other potential enhancements include:

- integration with other fielded or prototyped tactical equipment.
- creation of a more realistic JSTARS downlink to a Ground Station Module (GSM) using Emission PDUs.
- integration of an intelligence model into CLCGF.
- summarizing and reporting DIS indirect fire to Eagle so constructive model units can be attritted by DIS indirect fire
- full parsing of Eagle OPORDs and mapping into missions for deaggregated units
- investigating solutions to the terrain correlation issues between Eagle and ModSAF
- developing the standards necessary to distribute the "pseudo-deaggregation" capability to sensor models which are under the control of simulations other than ModSAF

- implementing a generic resolution management capability which will allow deaggregation requests to be registered and controlled by a single module.

The CLCGF's current role is to function as the central simulation engine executing a Korean scenario in the September, 1995, JPSD demonstration. Our future development will be guided by the needs of the JPSD program and other CLCGF users.

7. Conclusions

We have accomplished the work described in this paper in a relatively short period of time: CLCGF software design began in June, 1994; software development of the SIU, and various ModSAF modifications began in August, 1994; integration with Eagle began in November, 1994; and Eagle/SIU/ModSAF interoperation was first successfully demonstrated in the CLCGF system in January, 1995. Since the first demonstration, we have been improving the architecture and expanding the functionality of the CLCGF to support the long-term needs of various sites involved in the JPSD program.

We have created a CLCGF which can be used by the JPSD program to aid in the study, analysis, and testing of precision strike scenarios, and by the Depth and Simultaneous Attack Battle Lab at Ft. Sill for artillery system operator training and the study of new artillery weapons, systems, missions, and concepts at the Army Corps level. This has been accomplished by integrating constructive, virtual, live, and engineering simulations together into a usable, useful environment.

8. Acknowledgment

This work is being sponsored by STRICOM, the Topographic Engineering Center (TEC), and the Depth and Simultaneous Attack Battle Lab at Ft. Sill, under the Joint Precision Strike Demonstration program, contract number DACA76-93-D-0007, Delivery Order 3. We would also like to thank TRAC Ft. Leavenworth for their efforts in the development of Eagle scenarios and supporting the ModSAF integration effort in numerous ways.

9. References

Calder, R., Evans, A., "Construction of a Corps Level CGF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 4-6, 1994.

Calder, R., Peacock Jr., J., Wise, B., Stanzione, T., Chamberlain, F., Panagos, J. , "Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation,* Orlando, FL, May 9-11, 1995.

Hardy, D and Healy, M, "Constructive and Virtual Interoperation: A Technical Challenge",*Proceedings of the 3rd Conference on CGF and Behavioral Representation*, March 1993

Karr, C., Franceschini, R., Perumalla, K., Petty, M., "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation,* Orlando, FL, March 17-19, 1993.

Karr, C., Root, E., "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 4-6, 1994.

Smith, J. E., "Persistent Object Library Programmer's Guide", ModSAF 1.3 Software Documentation, Loral Advanced Distributed Simulation, Inc., Cambridge, MA, September, 1994.

"Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications, Version 2.0, Third Draft", Institute for Simulation and Training, Orlando, FL, May 28, 1993.

"A Modular Solution for Semi-Automated Forces, ModSAF, An Overview", Loral Advanced Distributed Simulation, Inc., Cambridge, MA, May, 1993.

10. Authors' Biographies

Robert B. Calder is a Senior Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in the development of DIS CGF systems for over four years, and is currently performing design and development on the ARPA Command Forces (CFOR) project. His primary research interests are in the area of tactics and behavior representation and generation for computer generated forces. Mr. Calder has a Master of Science degree in Computer Science from Boston University.

Jeffrey C. Peacock, Jr. is a Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in DIS CGF systems for the past year. Prior to entering the DIS CGF arena Mr. Peacock spent 7 years developing real-time embedded software systems. Mr. Peacock holds a Bachelors of Science degree in Computer Science from Merrimack College, Andover, MA.

James Panagos is a Consultant for TASC. He has been involved in the development of DIS CGF

systems for over 9 years, and is currently performing design and development on the ARPA Command Forces (CFOR) project. His primary research interests are in the area of tactics, behavior representation, automated planning and generation for computer generated forces. Mr. Panagos has a Master of Science degree in Computer Science from Massachusetts Institute of Technology.

Thomas E. Johnson is a Senior Software Engineer in the Electronic Systems Division at Raytheon Company. He has been involved in the development of real-time simulation models and manned flight simulators over the past thirteen years. His primary interests are in the area of man-machine interface and the development of tactical simulation models. Mr. Johnson has a Bachelor of Science degree in Aerospace Engineering from the Virginia Polytechnic Institute and State University, Blacksburg, VA.

Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF

Robert B. Calder, Jeffrey C. Peacock, Jr., Ben P. Wise
SAIC
486 Totten Pond Road
Waltham, MA 02154
rcalder@bos.saic.com, jpeacock@bos.saic.com, bwise@bos.saic.com

Thomas Stanzione, Forrest Chamberlain, James Panagos
TASC
55 Walkers Brook Drive
Reading, MA 01867
tstanzione@tasc.com, flchamberlain@tasc.com, jpanagos@world.std.com

1. Abstract

The integration of constructive and virtual simulation systems is a simulation research topic which has received much attention in recent years. The goal of this integration is to develop a virtual battlefield in which aggregate units and entities coexist, interacting with one another at a single level. This has typically been handled by requiring aggregate units to deaggregate to the entity-level when a detailed interaction is to occur. Even the most successful integrations to date, however, have only achieved limited success in minimizing the number of aggregations/deaggregations performed, and efficiently and accurately implementing the aggregation/ deaggregation process.

In this paper, we present work which has been performed on the JPSD CLCGF project with an emphasis on the implementation of a dynamic aggregation/deaggregation process which is efficient, accurate, and minimizes the number of aggregations/deaggregations performed.

2. Introduction

2.1 The JPSD Program

The Joint Precision Strike Demonstration (JPSD) program's goal is to introduce and implement new technologies into the defense arena that can address and correct precision strike deficiencies. To facilitate this goal, the JPSD program has created a simulation environment which is used to evaluate technologies, train users, and perform experiments necessary to reduce sensor-to-shooter timelines and to attack high-value, time-sensitive targets. As part of this environment, the JPSD program has sponsored the construction of the Corps Level Computer Generated Forces (CLCGF) system.

The primary purpose of the CLCGF is to provide the corps level simulation environment for DIS exercises in which the above mentioned program goals can be carried out. The CLCGF is used during the JPSD exercises to simulate maneuver and artillery units contained in an Army corps. The simulated units provide stimulus for and interact with tactical hardware systems and their operators, and interact with high-fidelity, engineering simulations. The CLCGF has been created by integrating the Eagle constructive simulation with the ModSAF entity-level simulation.

2.2 The CLCGF System

Entity-level simulations represent each entity which exists on the virtual battlefield at the individual platform level. They typically represent entities from the individual platform level up to the company level. They use the DIS protocol to interact with other entity-level simulations, and simulate the physical characteristics of each entity to determine battlefield outcomes. On the other hand, constructive simulations represent groups of entities as single, aggregate unit objects. They typically represent units at the company or battalion level up to the division or corps level. They are typically not designed to interact with other simulations, but instead simulate the entire battlefield internally, and use Monte-Carlo techniques to determine battlefield results.

The DIS environment has traditionally included only entity-level simulations. It has provided a sound environment for small-scale, tactical troop training, as well as a potential testbed for evaluating new vehicle and weapon systems. However, simulating the effects of entity-level simulations in corps level operations has remained beyond the reach of the DIS environment, due to network bandwidth and computer resource constraints. Using current network and

computer technology, a traditional DIS exercise is simply not capable of supporting a corps level operation. This was the primary motivation for creating a CLCGF which utilizes both constructive and entity-level simulations. Transmission of unit state data at the aggregate level is a key factor which decreases network load by significantly decreasing the number of PDUs transmitted in a large-scale exercise. If DIS is to support a 100,000 entity exercise, representation of some units on the battlefield as aggregates is likely.

The simulation engine of the CLCGF has been built by integrating the constructive, aggregate-level simulation Eagle, with the virtual, entity-level simulation ModSAF. This simulation engine interacts with various live, tactical hardware systems, including: the All Source Analysis System (ASAS) Warrior and Ground Station Simulator (GSS) for presentation of the tactical battlefield situation to the operator and potential target nominations; and the Automated Deep Operations Coordination System (ADOCS) for the creation and assignment of fire missions. The simulation engine interacts with the STRIKE engineering-level simulation, which simulates the deployment and flyout of smart submunitions. It also interacts with the TAFSM simulation, which it utilizes to simulate the deployment and flyout of various smart submunitions. In addition, the CLCGF interacts with various other DIS simulations, such as the Warbreaker SimCore simulation.

In order to allow military training and analysis of scenarios of interest to JPSD, the CLCGF must generate a full corps-level exercise. To accomplish this goal, many technical challenges need to be addressed. These involve issues such as efficient incorporation of aggregate units into DIS, effective incorporation of DIS entity-level information into constructive simulations, development of a dynamic aggregation/deaggregation protocol, interaction between constructive and entity-level simulations, and interaction between a constructive/virtual simulation, live systems, and engineering-level simulations. The work performed on the CLCGF to date has focused on these fundamental goals.

2.3 Constructive/Virtual Simulation Linkage

Bringing together constructive and virtual simulations on the synthetic battlefield creates difficult technical challenges. These difficulties lie in many areas, including management of hardware and network capacity across simulations, ensuring correlation of data and behaviors across simulations, resolving timing issues, ensuring terrain and environment correlation, and implementing an efficient, effective set of communication protocols between simulations. The JPSD CLCGF project has implemented solutions which begin to address many of these problems. The remainder of this paper will focus on modifications made in two key areas which are critical to implementing an accurate and efficient aggregation/deaggregation process:

Managing Capacity: All constructive/virtual linkage projects to date require that aggregate to entity interaction be performed at the entity level. However, it is not practical in terms of hardware and network capacity to decompose all aggregate units into virtual world entities whenever the two come within potential interaction range or enter a pre-defined area on the virtual battlefield, as this will lead to idle echelons occupying valuable computer resources and manpower, and spreading deaggregation. A dynamic aggregation/deaggregation process is needed, in which aggregate units deaggregate when a virtual world entity intends to interact with them, or they intend to interact with an entity. Using this philosophy, aggregations/deaggregations will only occur when absolutely necessary.

Managing Correlation: A key factor in successfully linking constructive and virtual simulations is ensuring data and behavior correlation between simulations. Anomalous results will be obtained from a linkage between systems which are poorly correlated. In order to eliminate these anomalies, a detailed mapping between the abstract and approximate definition of a constructive unit and the precise representation of entities in a virtual world unit is needed.

The CLCGF system addresses these issues within the context of the JPSD program. The CLCGF is composed of the Eagle aggregate-level, constructive simulation and the ModSAF entity-level, virtual simulation. The major development effort on the CLCGF project has been the implementation of the Simulation Integration Unit (SIU), whose purpose is to bring together the constructive world of Eagle and the virtual world of ModSAF.

The CLCGF architecture is described in more detail in these proceedings (Calder et. al. 1995) and will not be revisited here. The purpose of this paper is to present the innovative technologies devised in the CLCGF to deal with the issues of managing both capacity and correlation in a constructive/virtual simulation linkage. The following sections focus on each of these aspects separately.

3. Managing Capacity

3.1 Changing Simulation Resolution

When an aggregate unit deaggregates into its component entities, a handoff of simulation is made from the constructive to the virtual simulation. Similarly, when an entity-level simulation unit is re-aggregated into an aggregate unit from its component entities, a handoff of simulation is made from the virtual to the constructive simulation. We define the process of performing this handoff without significant loss of state as a resolution change. Other changes in fidelity during the course of a simulation, such as swapping vehicle dynamics models, constitute resolution changes, but are not modeled in the CLCGF. The term "resolution change" used throughout the remainder of this paper refers to aggregations and deaggregations only.

Current approaches to resolution change have defined static criteria to trigger aggregation and deaggregation, that is, criteria that do not change as the simulation evolves. The Integrated Eagle/BDS-D project (Karr et. al. 1993) defined one or more "high-resolution areas" in which constructive units are required to deaggregate into virtual entities. The BBS/DIS project (Hardy et. al. 1993) defined a "sphere of influence" for virtual entities, within which constructive units are required to deaggregate into virtual entities. In either case, constructive simulations relinquish control entirely to virtual simulations within these areas.

A problem resulting from the "high-resolution area" approach to resolution change is that often times needless deaggregations will occur for aggregate units which pass into the high resolution area, but do not interact with any entities over the course of the simulation. This wastes precious network bandwidth in large-scale simulation exercises. This needless deaggregation can be controlled by scripting the simulation scenario carefully to avoid needless deaggregation, but this questions the validity of the scenario.

A problem resulting from the "sphere of influence" approach to resolution change is that spreading deaggregation can easily result. Spreading deaggregation occurs in the following situation: an entity comes within interaction range of an aggregate unit, which causes the aggregate unit to deaggregate; when the aggregate unit deaggregates, one or more of its entities is within interaction range of another aggregate unit, which causes that aggregate unit to deaggregate; and this process continues on. Spreading deaggregation wastes both network bandwidth and computer resources, since there is no reason for any aggregate units other than the one within interaction range of the entity to deaggregate. It can easily spiral out of control and cause the entire battlefield to change resolution to the entity-level, thereby defeating the purpose of using the constructive model in the first place. This spreading deaggregation can be controlled by scripting the simulation scenario carefully to prevent it from occurring, but this clearly introduces an undesirable bias into the scenario.

In contrast, resolution changes in the CLCGF are dynamic, based upon events which occur during the exercise or upon initiation by a human operator at a GUI. The CLCGF does not rely upon predetermined high-resolution areas or spheres of influence to initiate resolution changes, but it does not preclude these mechanisms from being used. Instead, it places the responsibility for initiating resolution changes on the simulation which intends to interact with an entity or unit simulated in the other world. This requires that the entity-level simulations in an exercise be capable of receiving and processing aggregate unit state information to decide whether interaction with an aggregate unit is desired. This alleviates the problems of needless deaggregation and spreading deaggregation since only those units which intend to interact are deaggregated. Additionally, this facilitates scenario construction which is untainted by the mechanisms implemented in the underlying constructive and virtual simulations.

Re-aggregation in the CLCGF is initiated when the event which triggered the deaggregation is complete, or upon operator request (if the deaggregation was operator initiated). In any case, it is the responsibility of the simulation which requested the deaggregation to initiate the re-aggregation, since it has the context of why the deaggregation was needed initially. If another simulation requires that the deaggregation be maintained, then it assumes responsibility for maintenance of deaggregation until it is no longer required.

3.2 Resolution Change Protocol

A protocol for aggregate units and the initiation of resolution changes has been defined and implemented in the CLCGF. It specifies the format and the transmission, receipt, and processing requirements of DIS 2.0.3 experimental PDUs to facilitate aggregate units. Specifically, two new PDUs have been defined: the Aggregate State PDU (ASPDU) and the Deaggregation Request PDU (DRPDU).

The ASPDU is similar in usage and purpose to the DIS Entity State PDU, but is used only for aggregate units. It allows Eagle aggregate units to be broadcast on the DIS network similar to the way entities are normally broadcast in traditional DIS exercises. The ASPDU for each aggregate unit is transmitted by the SIU every five seconds and contains the following

fields: entity id, unit type, unit marking, aggregate state (i.e. aggregated or deaggregated), position, orientation, velocity, formation, extent, number of entities in the aggregate, and subordinate entity ID's (when the aggregate unit is in the deaggregated state).

All resolution changes in the CLCGF are managed by the SIU, since it is the link to the constructive simulation. Resolution changes can be initiated by the constructive simulation (Eagle), any virtual simulation on the DIS network (e.g. ModSAF), or the SIU (via event-driven mechanisms). Regardless of the source of the resolution change request, the same protocol is used to initiate, maintain, and terminate the resolution change.

To initiate the deaggregation of an aggregate unit, a simulation issues a DRPDU with the ID of the unit to be deaggregated. To maintain deaggregation of the unit, this DRPDU is retransmitted periodically (e.g. every five seconds). Re-aggregation of a deaggregated unit is initiated by ceasing transmission of DRPDUs for that unit. Re-aggregation will occur when no DRPDUs are received for a given unit for a period of 2.4 times the retransmission rate (e.g. 12 seconds). This timeout mechanism allows another simulation to take over responsibility for maintaining deaggregation (i.e. transmitting DRPDUs) if the originally responsible simulation terminates its interest (i.e. ceases sending DRPDUs).

3.3 Sources of Resolution Change in CLCGF

As described above, the CLCGF uses dynamic criteria for resolution change. Resolution changes are initiated and terminated based on the current tactical situation. The state of the resolution of all units in the simulation at a given time, therefore, is based solely on the sequence of events leading up to that time. The mechanisms currently implemented which trigger resolution changes include assignment of a fire mission to an MLRS unit (from live, tactical hardware or the ModSAF GUI), ModSAF operator request, SIU operator request, and Eagle request.

In a typical JPSD exercise, a hostile target nomination is received by the Automated Deep Operations Coordination System (ADOCS) operator. The operator creates a fire mission for that target, and a Full Mission Call For Fire (FMCFF) message, enclosed in a DIS Signal PDU, is dispatched to the SIU. Upon receipt, mission parameters are used to approximate the vehicles to be deaggregated (i.e. the fire support battery and any potential enemy targets in the target area. Receipt of the FMCFF message serves as the trigger mechanism for dynamic resolution changing in the CLCGF. Initiated by events derived from the exercise itself, it is the first example of an event-based resolution change used to date.

The SIU operator, as the controller of the exercise, is able to initiate a resolution change from the SIU's Graphical User Interface. For example, the SIU operator may decide that a certain engagement need be resolved at the entity level. This may be done by deaggregating the interacting units, which will transfer their control and modeling to ModSAF.

By use of a similar GUI, the ModSAF operator may change the resolution of any unit. This allows operators with a non-exercise-wide view to control the fidelity of the simulation, and change resolution based on a local decision criteria.

Eagle may specifically request a resolution change for any unit. This supports the "high resolution" interaction area approach of conducting virtual to constructive interactions. It is supported but, to date, has not been used in CLCGF.

3.4 CLCGF Architectural Support for Dynamic Resolution Change

The CLCGF's ability to efficiently and accurately perform dynamic resolution changes is made possible by three key design decisions: to tightly couple the SIU to ModSAF, to represent all of Eagle's constructive units in ModSAF's virtual world as aggregates, and to require that Eagle units be simulated at the company (or battery) level.

3.4.1 Tight Coupling of SIU to ModSAF

The SIU is tightly coupled to ModSAF. Like ModSAF, it utilizes the Persistent Object (PO) protocol to maintain the state of units, graphics, and behaviors. This allows the SIU the same full range of functions over an aggregate unit as if it were a ModSAF unit itself.

For example, creation of aggregate units in the SIU is initiated upon receipt of a Unit Create message from Eagle. The SIU translates the aggregate unit parameters into PO unit parameters for the given unit type, and then broadcasts a unit creation PO message. The SIU then begins simulation of the unit as a ModSAF unit, as though it had been created from the ModSAF GUI. The unit remains in the ModSAF PO database and is updated at the appropriate times.

Similarly, upon deaggregation of an aggregate unit, the SIU translates the unit's current state into PO unit parameters for each entity in the unit organization. It also creates a PO taskframe for the deaggregated unit

based on the aggregate unit's mission. The SIU then broadcasts unit creation and taskframe PO messages. A CLCGF ModSAF simulator which is on the same PO database ID as the SIU begins simulation of the entities in the unit, and execution of the taskframe, within ModSAF.

3.4.2 Representation of Eagle Units in the SIU and ModSAF

Aggregate unit state information is broadcast on the DIS network using the ASPDU defined above. In order to generate the information contained in the ASPDU, however, it is necessary to maintain Eagle aggregate unit state in the SIU. This is accomplished in the ModSAF SIU by handling aggregate units similar to the way in which ModSAF handles entities.

At scenario start, Eagle sends the initial state of each aggregate unit to the SIU, and the SIU creates a local aggregate simulation unit. This local aggregate unit is comprised of simply an aggregate hull, and is entered into the SIU's vehicle table. It ticks similarly to the way ModSAF entities do, but does not execute any ModSAF tasks, since Eagle controls its movement. This enables the SIU to simulate local aggregate units in the same manner as ModSAF simulates local entities.

The periodic transmission of ASPDUs by the SIU enables ModSAF, as well as other DIS simulations, to consider aggregate units as remote entities. We have modified ModSAF to receive and process the ASPDUs, and to incorporate remote aggregate units in the same manner that remote entities are incorporated. This enables the display of aggregate units on the ModSAF PVD, which allows the operator to see the entire battlefield of aggregates and entities, and to perform aggregate unit operations (e.g. aggregations and deaggregations) as described above. Processing of ASPDUs also enables aggregate units to be processed by ModSAF sensor models.

3.4.3 Eagle Unit Simulation at Company Level

ModSAF organizes entities in units up to the company level and supports company-level behaviors. Similarly, Eagle is capable of explicitly simulating units down to the company level and supports company-level behaviors. The mapping of data across the two systems can thus be made without significantly changing the content of the data. Had the two systems not met at a common echelon level, a new set of problems would have been introduced with respect to handling this gap.

To decompose an Eagle company into its constituent members in the virtual world, the SIU utilizes a user defined, run-time parameter file to map Eagle company compositions into ModSAF compositions. The unit type defines the formations it can assume, its mix of vehicles, and other significant tactical behaviors. Appropriate ModSAF units are chosen from a pre-defined list. The ModSAF unit types are not currently composed dynamically, but solutions are being explored.

Requiring that Eagle units be simulated at the company or battery level facilitates translation from Eagle missions to ModSAF missions when a change in resolution occurs. It also allows for ease in reporting state information of ModSAF deaggregated units to Eagle.

When an Eagle aggregate unit is deaggregated into ModSAF entities, the ModSAF unit is automatically assigned a ModSAF mission based upon its mission in Eagle. Operations orders (OPORDs) are passed from Eagle to the SIU upon deaggregation. However, the SIU does not currently parse the entire OPORD to automatically construct the ModSAF unit mission. Instead, the SIU uses the current operational activity of the Eagle unit, maps it to a ModSAF taskframe, creates the taskframe (using other information from Eagle such as heading, speed, formation, etc.), and assigns the taskframe to the deaggregated ModSAF unit. The ModSAF unit immediately begins execution of the taskframe. In the future, we plan to fully parse the OPORD to automatically construct and assign a consistent mission to the ModSAF unit.

3.5 A Generalized Resolution Management Solution

The CLCGF uses a variety of mechanisms to implement resolution change. Yet, in CLCGF as in other implementations, individual simulations or operators are burdened to initiate, monitor, and terminate resolution changes. A more general approach is desirable, where resolution control is analyzed and executed from a centralized module. This module will centralize resolution management requirements in one location (thereby avoiding code duplication) and reduce the overhead of communications. Resolution change requirements can be defined over the course of the exercise. Resolution changes themselves can occur at any time via a reliable resolution management protocol.

The generalized resolution module would control changes based on the following categories of criteria:

- ingress/egress of a high-resolution area. When a unit enters or exits a specified area, a resolution change occurs.

- ingress/egress of a sphere of influence. When a unit comes within a specified range of another unit, a resolution change occurs.

- time-based resolution changes. When a unit enters a window of time, defined in either absolute or relative terms, a resolution change occurs.

- event-based resolution changes. When a specific scenario event occurs, a resolution change occurs.

Using this approach, simulations which desire resolution changes based on the above criteria register their specifications with the resolution management module over the course of the simulation. These specifications consist of the categories listed above, and their logical combinations. The resolution management module monitors the simulation with respect to all registered resolution change specifications. When resolution change conditions are met, it initiates a resolution change, and maintains the resolution change until termination conditions are met.

4. Managing Correlation

4.1 Physical Characteristics

When linking constructive and virtual simulations, a correlation must be made between representations which are rooted in vastly different domains. Units in a constructive simulation do not maintain detailed information regarding their constituent entities. Therefore, when deaggregating an aggregate unit, an entity-level simulation must utilize the known physical characteristics of the unit. By utilizing this information, an accurate representation of the unit in the virtual world can be derived.

It is critical that the physical characteristics of an aggregate unit, such as unit composition, strength, location, heading, and formation, can be accurately reconstructed in the deaggregated unit's entities. In the CLCGF, some physical characteristics, such as unit composition, are specified in data files which are correlated between the SIU and Eagle. Others are transferred from Eagle to the SIU as state information during the course of the scenario. This ensures that these physical characteristics will be accurately mapped from the constructive to virtual, and vice versa, when a resolution change occurs.

4.2 Intelligent Entity Placement

When deaggregating a constructive unit, an entity level simulation must utilize the characteristics of the virtual environment and the mission which the unit is executing. By utilizing this information an accurate representation of the constructive unit in the virtual world can be derived, in which vehicles are placed at reasonable locations and headings.

We have developed software which makes use of the available mission information in the Eagle constructive simulation, as well as knowledge of the virtual environment's terrain, to generate vehicle placements in the virtual world in real-time. This software is utilized for both real and pseudo-deaggregation of Eagle units, which ensures correlation between entity placements of a pseudo-deaggregated unit and the same aggregate unit when it is actually deaggregated.

The intelligent placement of entities on the virtual battlefield is based on satisfying constraints which can be subdivided into two classes: mission independent constraints and mission dependent constraints.

4.2.1 Mission Independent Constraints

Mission independent constraints are applied during all deaggregations regardless of the constructive unit's mission (e.g. road march, occupy battle position, etc.) These constraints include the constructive unit's physical characteristics (e.g. unit composition, location, heading, and formation), as well as the characteristics of the virtual environment (e.g. obstacles, road networks, etc.). The physical characteristics of the constructive unit are used to template the individual entities which make up the unit. This templating process is accomplished using the inherent capabilities of ModSAF by defining new entries in the echelon database. These new entries coupled with location, formation, and heading information provide enough data to establish an initial lay down of the entities. The next step is to take into account the mission dependent constraints, which are discussed in the next section. Once the mission dependent data has been factored into the vehicle placement, the attributes of the underlying terrain must be taken into account. Adjustments are made to the vehicle position so that vehicles do not overlap each other or obstacles found on the terrain.

Checking individual vehicles for overlap with obstacles is necessary since constructive and entity level simulations have different representations of the underlying terrain. The Eagle terrain database is intended to support terrain reasoning by echelons at the company level and higher. During Eagle's terrain generation process the terrain features are aggregated to form go and no-go areas. These go and no-go areas are then used to define mobility corridors for the aggregate units. In addition, large obstacles, general area mobility, and intervisibility characteristics are computed for these areas. In contrast, ModSAF

utilizes a more detailed terrain database which is intended for use by individual vehicles. The information stored in ModSAF's terrain database includes road and river networks, small obstacles, and sampled elevation data. Thus, it is often the case that a clear region of terrain in Eagle actually contains obstacles that may impede the movement of individual entities in ModSAF. For example, a given location that is within a mobility corridor on one database may be in a densely forested area on the other. We must therefore check each individual entity for overlap with obstacles when deaggregating a constructive unit.

4.2.2 Mission Dependent Constraints

Incorporating mission dependent constraints during the deaggregation process forms the majority of the intelligent entity placement code added in support of the CLCGF program. These constraints are related to the specific mission which the unit is performing. Missions that have been identified as needing separate constraint sets include road march, attack, and defend.

A unit performing a road march mission attempts to travel from an initial location to a destination utilizing local road networks, while maintaining a column formation. In our current implementation, ModSAF is not given the destination. As a result we have implemented code that uses the initial vehicle placements, as described in Section 4.2.1, and the desired heading to perform intelligent entity placement on the road network. The process involves looking for a suitable road segment in the vicinity of the constructive unit's current position. The center of mass of the unit is then placed on the closest point of the selected segment and the unit is expanded outward from the center. Subsequent road segments are chosen for vehicles as they are created, using the direction of the initial segment as a filter criteria. Placement of a portion of the vehicles on roads and the remainder off the road is supported when the local road network is insufficient.

Figure 1 illustrates the deaggregation of a unit which is heading Southeast and executing a road march. The center of mass is first placed on the nearest point of segment C. The first half of the unit formation is then expanded forward along segment C and onto segment E. The second half of the unit is then expanded backward along segment C onto segment A. Since the road network ends with segment A, any remaining vehicles will be placed on the terrain as if segment A continued further to the Northwest.

The vehicle placement functionality is designed to work for both the simple case of vehicles in a column formation, and the more difficult case of vehicles in an arbitrary formation. For column formations, each vehicle is placed directly on the road and the desired inter-vehicle spacing is maintained. For non-column formations the width of the formation is maintained, the center of the formation follows the road, and inter-vehicle spacing is scaled based on the geometry of the turns in the road. The road following code is essentially independent of the underlying road

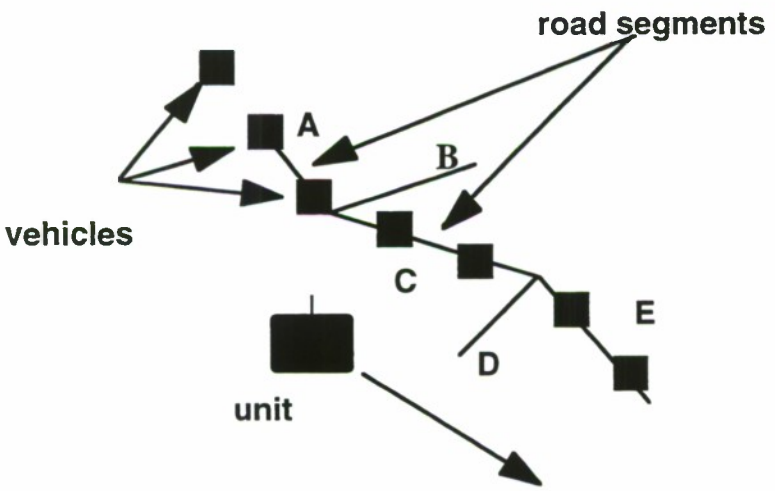


Figure 1: Intelligent Entity Placement for Roadmarch

representation. Thus, it can be used with other terrain database formats, and can also be used to generate vehicle placements that follow linear terrain features other than roads. For example, this could be used to place vehicles such that they follow a path between two points which maximizes cover. Implementation of placements for attack and defend missions have not been addressed to date. We anticipate that each will involve a mixture of cover, concealment, mobility, and line-of-sight constraints.

System response time to deaggregation requests is critical. Due to the frequency with which deaggregation is performed and the need for a smooth, rapid transition between constructive and entity-level simulations, it is critical that intellignet entity placement solutions emphasize performance while achieving realistic vehicle placement. We are therefore developing methods which rapidly approximate complex constraints. For example, a first-order attempt at maximizing cover might involve finding local terrain elevation minima.

5. Future Work

Future work will include expanding upon the current set of events which trigger dynamic aggregation/ deaggregation as well as improvements to the intelligent vehicle placement algorithms. Possible improvements in these areas include:

- Implementation of a generic resolution management library which will allow deaggregation requests to be registered and controlled by a single module. This will involve the definition of new criteria which are used to trigger the aggregation/deaggregation process, as well as an interface for specifying this criteria.

- Implementation of man-made obstacle avoidance algorithms in the entity placement process, to improve realism.

- Investigation of solutions to the terrain correlation issues between Eagle and ModSAF.

6. Conclusions

The development of new networking and computer technologies will certainly help increase the number of DIS entities which can participate in a large-scale exercise, but as technology increases so will the fidelity of the simulations. As fidelity increases, more resources are consumed and the net performance gain becomes insignificant. Dynamic aggregation/ deaggregation is one of the keys to supporting large-scale DIS exercises. Use of this scheme allows deaggregation to occur only when necessary, based on battlefield events or operator interaction. Computer and network resources are thereby conserved, as they are utilized by tactically significant units and not spent on tactically insignificant units.

The CLCGF system has successfully demonstrated dynamic aggregation/deaggregation, intelligent vehicle placement, and road network utilization in several demonstrations at the Integration and Evaluation Center (IEC) at the Topographic Engineering Center (TEC) located at Ft. Belvoir, VA.

7. Acknowledgment

This work is being sponsored by STRICOM, the Topographic Engineering Center (TEC), and the Depth and Simultaneous Attack Battle Lab at Ft. Sill, under the Joint Precision Strike Demonstration program, contract number DACA76-93-D-0007, Delivery Order 3. We would also like to thank TRAC Ft. Leavenworth for their efforts in the development of Eagle scenarios and supporting the ModSAF integration effort in numerous ways.

8. References

Calder, R., Evans, A., "Construction of a Corps Level CGF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 4-6, 1994.

Calder, R., Peacock Jr., J., Panagos, J. , Johnson, T., "Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation,* Orlando, FL, May 9-11, 1995.

Hardy, D and Healy, M, "Constructive and Virtual Interoperation: A Technical Challenge",*Proceedings of the 3rd Conference on CGF and Behavioral Representation*, March 1993

Karr, C., Franceschini, R., Perumalla, K., Petty, M., "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation,* Orlando, FL, March 17-19, 1993.

Karr, C., Root, E., "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 4-6, 1994.

Smith, J. E., "Persistent Object Library Programmer's Guide", ModSAF 1.3 Software Documentation, Loral Advanced Distributed Simulation, Inc., Cambridge, MA, September, 1994.

"Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications,

Version 2.0, Third Draft", Institute for Simulation and Training, Orlando, FL, May 28, 1993.
"A Modular Solution for Semi-Automated Forces, ModSAF, An Overview", Loral Advanced Distributed Simulation, Inc., Cambridge, MA, May, 1993.

9. Authors' Biographies

Robert B. Calder is a Senior Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in the development of DIS CGF systems for over four years, and is currently performing design and development on the ARPA Command Forces (CFOR) project. His primary research interests are in the area of tactics and behavior representation and generation for computer generated forces. Mr. Calder has a Master of Science degree in Computer Science from Boston University.

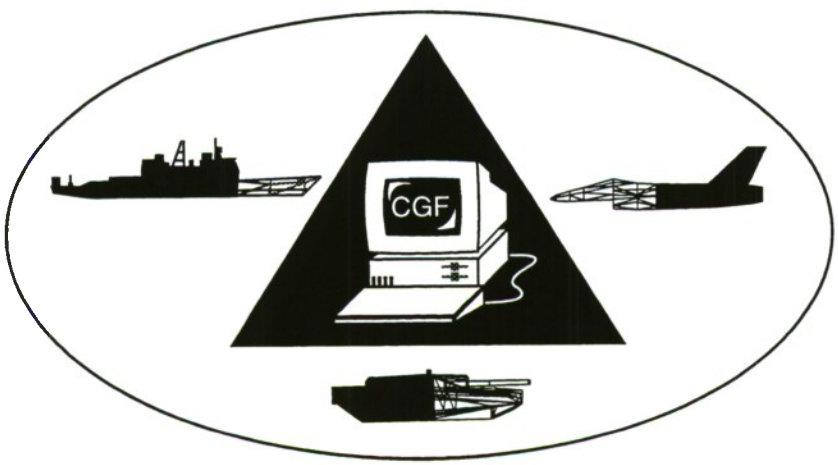
Jeffrey C. Peacock, Jr. is a Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in DIS CGF systems for the past year. Prior to entering the DIS CGF arena Mr. Peacock spent 7 years developing real-time embedded software systems. Mr. Peacock holds a Bachelors of Science degree in Computer Science from Merrimack College, Andover, MA.

Ben P. Wise is a Senior Scientist in the Technology Research Group at Science Applications International Corporation. He has twelve years experience in developing and implementing techniques for simulating intelligent behavior, including classical artificial intelligence techniques, game theory, decision analysis, C2 in constructive models, and CGF. Dr. Wise has a Bachelor of Science degree in Physics from the Massachusetts Institute of Technology, and a PhD in Engineering and Public Policy from Carnegie Mellon University.

Thomas Stanzione is the manager of the Synthetic Environment Section at TASC. He is the Program Manager for the ICTDB project and a key contributor to TASC's Synthetic Environment programs, including Weather in DIS (WINDS) and Multi-Echelon CFOR with ForeSight (MECFS). Prior to joining TASC, Mr. Stanzione served as the deputy director of the Semi-Automated Forces group at Loral Advanced Distributed Simulation (LADS). Mr. Stanzione has a Master of Science degree in Photographic Science from the Rochester Institute of Technology.

Forrest Chamberlain is a Member of the Technical Staff in the Signal and Image Technology Division at TASC. Mr. Chamberlain has been involved in Computer Generated Forces work since joining TASC in 1994. Prior to that, he was a critical contributor to the hardware and software design of a "wearable" computer system at Carnegie Mellon University, where he earned his Masters Degree in Electrical and Computer Engineering.

James Panagos is a Consultant for TASC. He has been involved in the development of DIS CGF systems for over 9 years, and is currently performing design and development on the ARPA Command Forces (CFOR) project. His primary research interests are in the area of tactics, behavior representation, automated planning and generation for computer generated forces. Mr. Panagos has a Master of Science degree in Computer Science from Massachusetts Institute of Technology.


CGF

Survey of Constructive + Virtual Linkages

David R. Stober, Matthew K. Kraus, William F. Foss, Robert W. Franceschini, and Mikel D. Petty
Institute for Simulation and Training
3280 Progress Drive, Orlando, FL 32826-0544
dstober@ist.ucf.edu

1. Abstract

This paper presents a survey of three projects at the frontier of constructive + virtual (C+V) linkages. C+V simulations solve problems that neither constructive nor virtual simulations solve well alone. Constructive simulation is frequently thought of as war gaming, simulating battle on high levels such as corps or division. Conversely, virtual simulation is at the vehicle level. Users battle in groups, using manned simulators or computer generated forces as friend and foe. The C+V linkage allows the commander to zoom in during a constructive battle to see action occurring at the vehicle level. C+V also allows players in a virtual training exercise to realistically participate in large scale scenarios. C+V supports execution of larger simulations than are possible on even the newest virtual simulations.

This paper will define C+V simulation integration and describe its importance in battlefield simulation. We will examine the following three C+V linkages: Integrated Eagle/BDS-D (the first project to integrate constructive and virtual simulations), Corps Level CGF (CLCGF), and BBS/SIMNET. This paper answers the following questions: How well do these integrations allow interaction across the C+V interface? How are aggregation and disaggregation handled? How are direct and indirect fire supported? What are some of the unsolved problems?

2. Introduction

2.1 Mission Statement

Simulation technology has seen steady growth for more than 10 years in both the areas of constructive and virtual simulation. More recently these technologies have been joined to solve new problems. This paper surveys how constructive and virtual simulations have been combined and what new problems they are solving. In the pages that follow we will discuss the state-of-the-art in constructive + virtual simulation.

2.2 Scope of the Survey

Many projects have researched and/or implemented integrations of constructive and virtual simulations. In this survey, we focus on systems in which aggregation and disaggregation are performed across the constructive and virtual boundary; such systems are termed "constructive + virtual linkages", or C+V linkages. Each project shown in Table 1 meets this requirement.

Project	*Constructive*	*Virtual*	*CGF*
Eagle/BDS-D	Eagle	DIS/SIMNET	IST CGF Testbed
CLCGF	Eagle	DIS	ModSAF
BBS/SIMNET	BBS	DIS/SIMNET	SIMNET/SAF

Table 1: C+V Projects

We will examine each of these C+V projects and evaluate them against a set of problem areas. For an introduction and tutorial on C+V linkages see (Franceschini 1995).

2.3 Report Organization

This paper has three key sections. Section 3 defines what constitutes a C+V linkage and its importance. Section 4 presents a review of current C+V linkages. Section 5 describes common problems found in building a C+V linkage and introduces some interesting C+V ideas.

3. A Characterization of C+V Linkages

3.1 Constructive Simulation

For the purposes of this discussion, we will use the following characterization of constructive simulations.

Constructive simulations represent military units (e.g., a tank company) as an aggregate without simulating each individual entity (e.g., tank) within the unit. The position, movement speed and direction, status, and composition of an aggregate unit are maintained for the unit as a whole, and are often computed as the result of statistical analysis

of the unit's actions. BBS, CBS, and Eagle are examples of constructive simulations.

3.2 Virtual Simulation

Virtual simulations represent each vehicle or fireteam as a distinct simulation entity. All necessary state information for each entity is maintained for that entity. Each entity is capable of independent action, and combat results are resolved at the entity level. The position, movement speed and direction, status, and composition of a military unit in a virtual simulation, if needed, must be inferred from the individual vehicles that compose that unit. SIMNET, BDS-D, and CCTT are examples of virtual simulations. A virtual simulation's entities may be controlled by either crewed simulators or computer generated forces (CGF).

3.3 Differences Between Constructive and Virtual Simulations

To summarize, constructive simulations represent aggregate military units while virtual simulations represent individual vehicles or soldiers as entities.

In general, constructive and virtual simulations differ in their treatment of time and space. In the case of time, virtual simulations usually intend that the apparent passage of time within the virtual environment of the simulation match that of the real-world that is being modeled; hence they are described as *real-time*. In contrast, constructive simulations are often time-stepped, with the simulation time advancing a fixed amount of time for each computational cycle of the simulation model. The size of the simulated time step ordinarily has nothing to do with the time required to compute the events of that time-step, so such simulations are not real-time.

As for space, virtual simulations often specify the terrain of the virtual environment in great detail, with individual roads, buildings, trees, and bushes represented. The possible locations a virtual entity may occupy are essentially continuous over the terrain. This detail is appropriate for entity level modeling. Constructive simulations generally use terrain that has been partitioned in a regular grid of squares or hexagons, with the terrain of each grid element abstracted into one or more terrain attributes that apply to the entire element (e.g., forest).

Finally, the traditional users of each class differ. Although there is considerable overlap, constructive simulations have historically been developed and used primarily by the analytic community to perform system and force development studies, whereas virtual simulations have been developed and used by the training community as training tools. Each community is learning to appreciate the advantages of the other class of simulation, and each is becoming more interested in using the best features of both classes.

Note that the term "unit" is used when referring to aggregates in a constructive simulation and the term "entity" is used when referring to vehicles or infantry in a virtual simulation.

3.4 Constructive+Virtual (C+V) Simulation

A constructive+virtual simulation is a system that links a constructive simulation with a virtual simulation. The goal of a C+V simulation is to have events in one simulation influence or effect events in the other simulation. Units or entities that are present in a constructive or virtual simulation are usually represented in some fashion in the other simulation.

4. C+V Integration Review

The following presents a review of Integrated Eagle/BDS-D, Corps Level Computer Generated Forces (CLCGF), and BBS/SIMNET. For each project, the following are described:

- The goal of each project
- Conceptual C+V configuration
- Description of the constructive system
- Description of the virtual system
- The interface between constructive and virtual simulations
- Interoperability between constructive and virtual worlds

Information about each project was gathered from the referenced papers as well as surveys sent to authors of the papers.

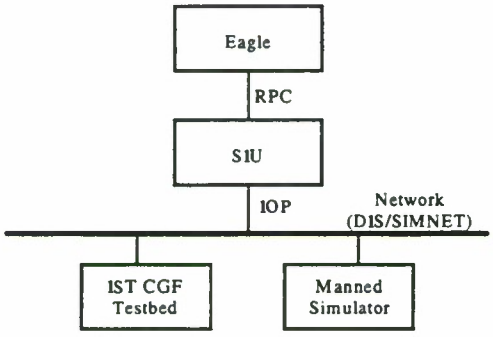
4.1 Integrated Eagle/BDS-D

The primary goal of the Integrated Eagle/BDS-D project is to integrate the Eagle constructive simulation (developed by the US Army TRADOC Analysis Center) with a DIS/SIMNET virtual environment using the Institute for Simulation and Training's Computer Generated Forces Testbed

(IST CGF Testbed). The project's objective is to prove by demonstration the concept of interoperability of constructive and virtual simulations.

4.1.1 Conceptual C+V Configuration

Figure 1 shows the conceptual system configuration. The Eagle simulator (see Section 4.1.2) and the SIU (see Section 4.1.4) communicate using Remote Procedure Calls (RPCs). Communication between the SIU and the IST CGF Testbed is done with an interoperability protocol (IOP). Some currently active IOP PDUs describe the composition of a unit, a change in a unit's status, a unit's Operations Order, and indirect fire between the constructive and virtual simulations.



LEGEND:

Nodes	Definition
Eagle	TRAC Eagle Constructive Simulator
SIU	Simulator Interface Unit
Manned Simulator	Manned Simulator
IST CGF Testbed	Virtual CGF System

Figure 1: Eagle/BDS-D Conceptual C+V Configuration (Karr 1994)

4.1.2 The Eagle Simulator

The Eagle simulator is a Corps and Division level combat model. The smallest units simulated are at the company and battalion levels. The Eagle simulator is used to analyze combat development studies, the effects of weapon systems, command and control, military doctrine, and organization on force effectiveness (Powell 1993). It was developed to provide an enhanced representation of command and control to and reduce the turn-around time of scenario development. The following functions are performed by Eagle in the Eagle/BDS-D system (Karr 1994).

- Simulate all aggregate units.
- Optionally send disaggregation/aggregation requests when units move inside/outside a "high-resolution area".
- Switch to real-time execution when any unit is disaggregated.
- After disaggregating a unit, send an Operations Order to the disaggregated unit.
- While a unit is disaggregated, update the unit's status and position by processing information received from the IST CGF Testbed.
- Respond to requests for indirect fire.

4.1.3 The IST CGF Testbed

The IST CGF Testbed connects to a SIMNET or DIS network and provides a mechanism for testing CGF control algorithms. The IST CGF Testbed runs on IBM PC-compatible computers. The IST CGF Testbed consists of an Eagle Manager, one or more Operator Interface (OI) computers (used as a console for the human operator), and one or more simulators (which control the behaviors of the simulated vehicles). The IST CGF Testbed has the following responsibilities (Karr 1994):

- When a disaggregation request for a unit is received, break the unit down to single entities.
- Upon disaggregation, use a vehicle placement algorithm to place vehicles around obstacles in the virtual terrain.
- Simulate the individual entities.
- Respond to a CGF operator's commands.
- Send Eagle Operations Orders to the OI controlling the disaggregated unit.
- When an aggregation request is received, remove proper entities from the virtual simulation.

4.1.4 The Simulation Interface Unit (SIU)

Developed by the Los Alamos National Laboratory (LANL), the SIU coordinates the communication between the Eagle simulator and the IST CGF Testbed. The SIU is SIMNET and DIS compatible. The SIU has the following responsibilities (Karr 1994):

- Synchronize Eagle's time-stepped simulation with the virtual real-time simulation.
- Update Eagle with events that have occurred in the virtual simulation.
- Perform terrain coordinate conversions between Eagle and DIS/SIMNET.
- Determine each disaggregated unit's "center of mass".
- Update the status of Eagle's disaggregated units by listening to appearance PDUs from DIS/SIMNET.

- Send aggregate unit information to the virtual world.

4.1.5 Appearance of Units in the Virtual World

At regular intervals, Eagle sends descriptions of aggregate units in Unit Detail PDUs (UDPDU). Each UDPDU contains a field for the status of the unit. The status may be *disaggregated, aggregated* (shown as an icon), *pseudo-disaggregated*, or *invisible*. The CGF Operator chooses how constructive units should appear.

A disaggregated unit is controlled in the virtual world. Individual entities are either controlled by the IST CGF Testbed or manned simulators.

Icon unit status is the lowest level of detail for an aggregate unit. An internal protocol aggregate PDU is sent for each unit in icon status. This status allows nodes on the network to display an icon for the aggregated unit. This minimizes network traffic while allowing nodes in the virtual world to be aware of units being simulated by Eagle (Karr 1994).

Pseudo-disaggregation is a more detailed level of unit appearance. An appearance PDU is produced for each vehicle within the unit at regular intervals every five to ten seconds. Locations of the pseudo-vehicles in the unit are based on the formation of the unit. This allows nodes on the network to see a formation of vehicles moving across the terrain. Since the unit is controlled by Eagle, pseudo-vehicles are not simulated as entities in the virtual world. Thus, pseudo-vehicles may not fire their weapons, sight other entities, or receive fire. This permits many entities to be placed in the virtual world, creating a realistic picture for sensor systems (Karr 1994).

Invisible unit status is used when an operator chooses to hide the aggregated unit from the virtual world. This is usually done when the virtual world is cluttered with aggregate units.

4.1.6 Interactions Across C+V Boundary

Test scenarios have been used to demonstrate the interoperability of the Eagle/BDS-D project. Typical scenarios have had operators initiate disaggregation of a unit. Aggregate units on Eagle disaggregate to CGF or manned entities in DIS/SIMNET. Eagle executes in real-time while any unit is disaggregated and processes DIS/SIMNET events. Aggregate units on Eagle may attack virtual vehicles in DIS with indirect fire. Combat occurs between the constructive and virtual worlds in the following ways:

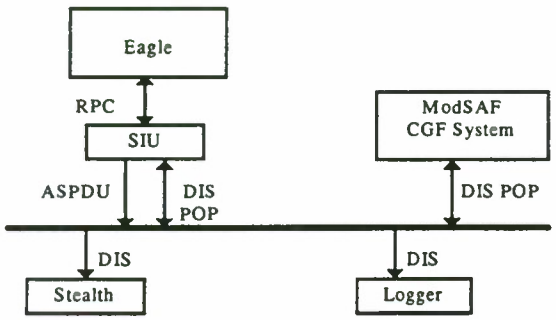
- When a Call for Fire is made from the IST CGF Testbed, the Eagle system responds with indirect fire. Indirect fire appears in the virtual world and damages virtual vehicles
- A disaggregated artillery battery with a Battery Fire Mission may send indirect fire at a constructive unit on orders from Eagle
- Operations Orders (unit missions) and Operator Intent (commands from a CGF operator) messages are transferred between Eagle and the CGF Operator

4.2 Corps Level CGF

The Joint Precision Strike Demonstration (JPSD) has a requirement to simulate a large number of entities on the DIS network. To satisfy this requirement, Corps Level Computer Generated Forces (CLCGF) was initiated to examine theater level simulations in DIS (Calder 1994). CLCGF integrates the Eagle constructive simulator with a DIS virtual simulation (using ModSAF). CLCGF links the constructive and virtual worlds by using a Simulation Interface Unit. This information about CLCGF was collected from (Calder 1995) (Calder 1994).

4.2.1 Conceptual C+V Configuration

Figure 2 illustrates the conceptual arrangement of the CLCGF system. For simplicity, only the constructive + virtual link is shown. The Eagle simulator and the SIU communicate using Remote Procedure Calls (RPCs). Communication between the SIU and ModSAF is done using Persistent Object Protocol (POP) (developed for ModSAF). The SIU also listens to DIS from the virtual simulation. Previous sections discussed the Eagle Simulator.



LEGEND:

Protocol	Definition
POP	ModSAF Persistent Object Protocol
RPC	Remote Procedure Calls
ASPDU	Aggregate State PDU

Nodes	Definition
Eagle	TRAC Eagle Constructive Simulator
SIU	Simulation Interface Unit
ModSAF	Virtual CGF System

Figure 2: CLCGF Conceptual C+V Configuration (after Calder 1995)

4.2.2 ModSAF

ModSAF is a well-known CGF system. For a description of ModSAF see "ModSAF User Manual Version 1.3" (Loral Advanced Distributed Simulation 1994).

4.2.3 The Simulation Interface Unit

CLCGF's SIU has the same responsibilities as the Eagle/BDS-D's SIU as listed in Section 4.1.4. To send aggregate unit information to the virtual world, CLCGF's SIU uses a modified version of the experimental DIS Aggregate Protocol, called the Aggregate State PDU (ASPDU). Aggregate State PDUs are similar to Entity State PDUs, but adapted for aggregate units.

4.2.4 Interoperabilities

Disaggregation is performed when a Disaggregate Request PDU (DRPDU) is sent to the SIU. A DRPDU must be resent periodically to keep the unit disaggregated. When the SIU initially receives a DRPDU, the SIU sends a POP message to ModSAF to disaggregate the unit. If a DRPDU is not received over a specific time period, the SIU will send a POP message to ModSAF to aggregate the disaggregated unit. Cease in transmission of DRPDU may be done by a ModSAF operator or by a timer.

CLCGF implemented indirect fire from the constructive world to the virtual world. When Eagle sends an indirect fire message to the SIU, the SIU broadcasts Detonation PDUs to DIS. DIS entities may be affected by the indirect fire.

As the final step in disaggregating a unit, Eagle sends an Operations Order to the SIU. The SIU is able to interpret a small portion of an Operations Order. For example, the SIU may interpret from an Operations Order a change in unit formation. The SIU sends commands in POP messages to ModSAF for execution of the Operations Order.

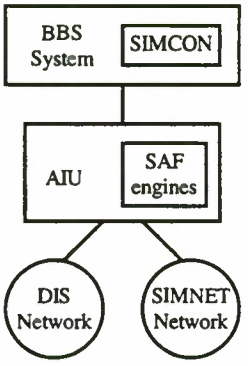
As stated above, the SIU broadcasts ASPDUs to the virtual world. Thus, all nodes in the virtual world have access to information about each aggregate unit. If a particular virtual node requires entity level information from an aggregate unit, the virtual node may internally pseudo-disaggregate the unit. This technique of pseudo-disaggregation places no Entity State PDUs on the network. For example, a ModSAF aircraft may contain a JSTARS system simulation. The JSTARS simulation internally pseudo-disaggregates the units that are within range of the radar. A vehicle placement algorithm is used to place the pseudo-disaggregated vehicles around obstructions on the virtual terrain.

4.3 BBS Linkage

The primary goal of the BBS linkage is to integrate a constructive simulation with a DIS network. The developers chose the Brigade/Battalion Battle Simulation (BBS) as the constructive simulation, because it can recognize and display individual vehicles on its graphics terminals. This information about the BBS linkage was collected from (Hardy 1994).

4.3.1 Conceptual C+V Configuration

As seen in Figure 3, the main linkage between BBS and the virtual world is the Advanced Interface Unit (AIU). The AIU communicates with both DIS and SIMNET networks. It also communicates with the BBS system through the Simulator Control (SIMCON). The SAF engines (see section 4.3.4) are part of the AIU. Since DIS CGF simulators were not available at the time the BBS project was started, SIMNET simulators are used to create virtual entities to interact with the constructive units.



LEGEND:

Nodes	Definition
BBS System	BBS Constructive Simulator
AIU	Advanced Interface Unit
SIMCON	Simulator Control
SAF engines	Virtual CGF Simulator

Figure 3: BBS Conceptual C+V Configuration

4.3.2 The Simulator Control (SIMCON)
SIMCON is a node on the BBS network that allows the AIU to access and control the internal workings of BBS. When the AIU's SAF engine is modeling the entities of a disaggregated unit at the virtual level, SIMCON stops BBS from modeling that unit. When the unit is reaggregated, SIMCON tells BBS to resume modeling the unit.

4.3.3 The Advanced Interface Unit (AIU)
The AIU has several components to complete the integration. The AIU includes several SAF engines to model the individual entities of disaggregated units. It also includes a DIS/SIMNET translator. This allows the AIU to recognize both DIS and SIMNET protocol and translate them so both worlds can recognize and interact with each other.

4.3.4 The SAF Engine
The SAF engine is a part of the AIU that models disaggregated units at the entity level in the SIMNET world. It recognizes a number of control functions received from the AIU. Some functions of the SAF engine are creating objects, deleting objects, and moving objects.

4.3.5 Interoperabilities
The AIU aggregates DIS entities into BBS objects, and periodically updates them so that the BBS simulator can see the virtual entities and interact with them. The AIU disaggregates BBS objects into SIMNET entities controlled by the SAF engine. The AIU also sends status reports back to the BBS simulator. For example, formation and damage assessments on a disaggregated unit would be transferred to the BBS simulator.

5. Findings and Observations

Concepts such as aggregation, disaggregation, and pseudo-disaggregation will be discussed in this section as well as the complications involved in these tasks. Also, some other problems and concepts of C+V simulation will be addressed.

5.1 Concepts

5.1.1 Disaggregation
Disaggregation is the mechanism by which control of a unit is transferred from the constructive simulation to the virtual simulation. When the unit is transferred to the virtual simulation it must be split into its individual entities. The virtual simulator that controls these entities is often a CGF system. Disaggregation is thus a one-to-many transformation.

Disaggregation involves the following steps:

1. Disaggregation is triggered.
2. Virtual simulator receives aggregate unit information.
3. Virtual simulator separates the unit into individual entities.
4. Virtual simulator creates each entity in the virtual world. Since more information is needed than is available, the virtual simulator must provide it.
5. Constructive simulation receives acknowledgment of disaggregation.
6. Constructive simulation releases control of the unit.

5.1.2 Aggregation
Aggregation is the mechanism by which control of a unit is transferred from the virtual simulation to the constructive simulation. When the unit is transferred to the constructive simulation, its vehicles are replaced by a single aggregate unit. Aggregation is thus a many-to-one transformation.

Aggregation involves the following steps:

1. Aggregation is triggered.
2. Constructive simulator receives aggregate unit information. Since more information is available than is needed, some information is discarded.
3. Virtual simulator removes the entities in the unit from the virtual world.
4. Constructive simulation resumes control of the unit.

5.1.3 Pseudo-Disaggregation

Pseudo-disaggregation allows a virtual simulation to display aggregate units as individual entities, but control remains with the constructive simulation. Pseudo-disaggregation is very similar to disaggregation, except that the individual entities of the unit are not modeled at the virtual level. Since the pseudo-disaggregated units are modeled at the constructive level, they will not respond to any interactions in the virtual world.

In Integrated Eagle/BDS-D, the Eagle Manager pseudo-disaggregates a unit, and sends Entity State PDUs for each vehicle in a unit. This allows each vehicle to be visible in DIS, while computation overhead is at a minimum (Karr 1994). In CLCGF, each node on the virtual network is responsible for internally pseudo-disaggregating units about which it wants entity level information. This minimizes network traffic (Calder 1995).

5.2 Problems

In each of the problems that follow, the designer is faced with temporal, spatial, or control problems. Additionally, the C+V integration must work with and sometimes extend rules of both the constructive and virtual worlds.

5.2.1 C+V Equal and Level Playing Field

Entities and units must have an equal and level playing field in which to simulate battle. The outcome of a battle between two constructive units should be similar to the outcome of that same battle held in the virtual world. Since aggregation is an *information loss* process and disaggregation is an *information gain* process, enough information should be retained in the aggregation/disaggregation cycle to insure similar outcomes. Since it is not feasible to keep all the information, we determine what information is most pertinent. Some questions that might help determine important information are:

- Can units and entities at less than full strength have multiple representations? A constructive unit of 6 helicopters at 50% strength may also be equivalent to 3 virtual helicopters at 100% strength.
- Should damaged entities appear undamaged after an aggregation/disaggregation cycle?
- Should entities low on fuel or payload be automatically refueled and reloaded after an aggregation/disaggregation cycle?

5.2.2 Spreading Disaggregation

One goal of C+V simulation is to keep network traffic and computation load to a minimum. Spreading disaggregation is a situation where a single disaggregation triggers a chain reaction of disaggregations that may include hundreds or thousands of DIS entities (Petty 1995) (Trinker 1994). For example: Four red heavy armor units and four blue heavy armor units form two lines across a front, in a constructive battle.

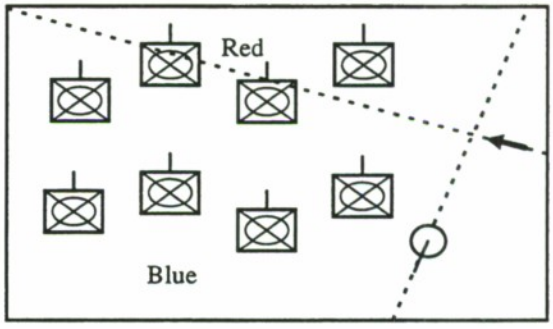


Figure 4: Spreading Disaggregation

A blue helicopter on a scouting mission passes within sensor range of a red armor unit. If the automatic disaggregation method used is close proximity, the red unit disaggregates. Next, a blue armor unit is disaggregated because it is in close proximity of a red virtual entity. The chain reaction continues until all eight constructive units have disaggregated. The passing helicopter may not have intended to interact with any of these units, yet all eight units have been dragged into the virtual world.

Now consider a blue missile (the arrow in Figure 4) launched from one corner of the battlefield to the opposite corner of the battlefield, in close proximity to red forces. Should all red forces along the path disaggregate? If red units can interact with the missile, should they disaggregate and engage? If the missile is traveling at supersonic speeds, the window of engagement is very short; how quickly should units disaggregate?

5.2.3 Unit Formation on Disaggregation

Disaggregation requires a transfer of control from a constructive unit into multiple entities in the virtual world. With this transfer of control, the virtual simulation has a location and orientation for each entity in a constructive unit. Typically these entities are ordered in a formation dependent on the Operations Order. For example, when the Operations Order is Road March, the entities will

be placed into a column formation. The disaggregation process involves solving several problems (Clark 1994), (Franceschini 1992):

- Entities must be placed intelligently onto the terrain. Most constructive simulations will not have the resolution for intelligent vehicle placement in the virtual terrain. For example, locations of trees, lakes, and rivers can impede realistic vehicle placement. Certain formations have strict line of sight requirements that are unavailable at the constructive level.
- If the command vehicle of a company is destroyed, a new vehicle must move into that position (physically and operationally) and assume those responsibilities.
- A formation of vehicles en route must perform dynamic obstacle avoidance. They cannot run into each other or fixed objects.

5.2.4 Direct and Indirect Fire

Direct fire across the C+V boundary is difficult to support, due to immense problems with both timing and database correlation (Trinker 1994). Constructive level simulations typically shift to real-time by leaping ahead in large time steps and then waiting for the virtual world to catch up. One direct fire exchange in the constructive world might be statistically calculated and executed in a fraction of a second. In the virtual world in order to be visually realistic, each phase of the direct fire must follow the rules and timing of physics. The constructive database can also be in an entirely different coordinate system and is normally at a much lower resolution.

Indirect fire across the C+V boundary is a difficult but solvable problem. Indirect fire from the constructive to virtual world works as follows: An Indirect Fire Volley is initiated against a ghost unit in the constructive world by selecting a munitions type and location. This is translated into a series of Detonation PDUs in the virtual world. Information on damaged or destroyed entities is then captured and relayed back to the constructive simulation (Franceschini 1995). Indirect fire from virtual to constructive is done when a disaggregated artillery unit chooses to attack an aggregated unit. Information about the attack (such as the locations of the fire detonations) is passed to the constructive simulation. The constructive simulation calculates statistical damage of the aggregate units near the fire detonations (Franceschini 1995).

6. Projects Outside the Scope of this Survey

This survey focused on projects that aggregate and disaggregate units across the C+V boundary. Some projects that researched C+V simulations implemented aggregation and disaggregation inside the constructive simulation. These projects employed a constructive simulation that supports both aggregate units and single entities. Some examples of these projects are IRIS, Janus, and SOFNET-JCM.

IRIS (Internetted Range Interactive Simulation) links constructive, virtual, and live simulators in a DIS environment. A primary goal of this project was to minimize the modifications to each simulator (Kazarian 1994).

Janus is a self-contained constructive simulation with an interface to DIS. Interfacing to DIS is managed through a Cell Interface Unit (CIU) called the World Modeler (Pratt 1995) (Pratt 1994).

SOFNET-JCM project generates interactions between the constructive Joint Conflict Model (JCM) Simulation and the SOF Inter-Simulator Network (SOFNET) aircraft simulator (Babcock 1994).

7. Conclusion

The evaluated C+V integrations have shown impressive results in this early stage of development. A long term goal is to provide a seamless interconnection supporting aggregation, disaggregation, and full interaction between the constructive and virtual simulation worlds. Current unsolved problems such as spreading disaggregation and the unequal and unlevel playing fields prohibit a fully automated solution. The current DIS standard 2.0.4 does not fully define an "Aggregate PDU" that will satisfy needs of the evaluated C+V integrations. However, C+V has proven to be an effective solution for large simulations.

8. Acknowledgment

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command (STRICOM) as part of the Signal Intelligence/Electronic Warfare project (contract N61339-93-C-0091). IST's ongoing work in constructive+virtual simulation is sponsored by STRICOM and the US Army TRADOC Analysis Center as part of the Integrated Eagle/BDS-D

project, contract number N61339-92-K-0002. That support is gratefully acknowledged.

9. References

Babcock, CDR D.B., Molnar, Maj. J.M., Selix, Maj. G.S., Conrad, G., Castle, M., Dunbar, J., Gendreau, S., Irvin, T., Uzelac, M., and Matone, J. (1994) "Constructive to Virtual Simulation Interconnection for the Sofnet-JCM Interface Project", *Proceedings of 16th Interservice/Industry Training Systems and Education Conference*, Orlando FL, June 1994, section 4-6.

Calder, R.B. and Evans, A.B. (1994) "Construction of a Corps Level CGF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 487-496.

Calder, R.B. (1995) Response to IST's Constructive/Virtual Survey, unpublished, March 7 1995.

Clark, K.J. and Brewer, D. (1994) "Bridging the Gap Between Aggregate Level and Object Level Exercises", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 437-442.

Franceschini, R.W. (1992) "Intelligent Placement of Disaggregated Entities", *Proceedings of the 1992 Southeastern Simulation Conference, Pensacola FL*, Oct. 22-23 1992, pp 20-27.

Franceschini, R.W. and Petty, M.D. (1995) "Linking Constructive and Virtual Simulation in DIS", *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing & Control and Dual-Use Photonics*, Orlando FL, April 17-21 1995.

Hardy, D. and Healy, M. (1994) "Constructive & Virtual Interoperation: A Technical Challenge", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 503-507.

DIS Steering Committee (1994) "The DIS Vision: A Map to the Future of Distributed Simulation" Version 1, *Technical Report IST-SP-94-01*, Institute for Simulation and Training, May 1994.

Karr, C.R., Franceschini, R.W., Perumalla, K.R.S., and Petty, M.D. (1993) "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 17-19 1993, pp 231-239.

Karr, C.R. and Root, E. (1994) "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 425-435.

Karr, C.R. and Root, E. (1994) "Integrating Constructive and Virtual Simulations", *Proceedings of 16th Interservice/Industry Training Systems and Education Conference*, Orlando FL, June 1994, section 4-5.

Kazarian, J.P. and Shultz, M.A. (1994) "The IRIS Architecture: Integrating Constructive, Live, and Virtual Simulations", *Proceedings of 16th Interservice/Industry Training Systems and Education Conference*, Orlando FL, June 1994, section 4-4.

Loper, M.L. and Petty, M.D. (1993) "Computer Generated Forces at the DIS Interoperability Demonstration", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 17-19 1993, pp 155-167.

Loral Advanced Distributed Simulation (1994) *ModSAF User Manual Version 1.3*.

Petty, M.D. and Franceschini, R.W. (1995) "Disaggregation Overload and Spreading Disaggregation in Constructive + Virtual Linkages", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 9-11 1995.

Powell, D.R. (1993) "Eagle II: A Prototype for Multi-Resolution Combat Modeling", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 17-19 1993, pp 221-230.

Pratt, D.R. (1995) Response to IST's Constructive/Virtual Survey, unpublished, February 13 1995.

Pratt, D.R., Johnson, CPT M., USA, and Locke, J., NPS (1994) "The Janus/BDS-D Linkage Project: Constructive and Virtual Model Interconnection", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 443-448.

Root, E.D. and Karr, C.R. (1994) "Displaying Aggregate Units in a Virtual Environment", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 4-6 1994, pp 497-502.

STOW-E 94 (undated) "Review of the Synthetic Theater of War - Europe"

Trinker, A. (1994) "General Architecture for Interfacing Virtual and Constructive Simulations in DIS Environment", *Technical Report IST-TR-94-28*, Institute for Simulation and Training, September 14 1994.

10. Author's Biographies

David R. Stober is a Research Assistant on the Integrated Eagle /BDS-D project at the Institute for Simulation and Training. He has earned a Bachelor of Science in Computer Science from the University of Central Florida. He is currently pursing a Master of Science degree in Computer Science from UCF. His research interests are in the areas of simulation and artificial intelligence.

Matthew K. Kraus is a Software Engineer at the Institute for Simulation and Training. Mr. Kraus has a Bachelor of Science degree in Computer Science from Western Michigan University. He is currently working on a Master of Science degree in Simulation Systems. His research interests are in the areas of distributed computing, artificial intelligence, and visualization.

William F. Foss is a Research Assistant on the Integrated Eagle/BDS-D project at the Institute for Simulation and Training. He is an undergraduate student in Computer Science at the University of Central Florida. His research interest is in the area of simulation.

Robert W. Franceschini is a Principal Investigator at the Institute for Simulation and Training. He currently leads the Integrated Eagle/BDS-D project at IST. Mr. Franceschini has earned a Bachelor of Science in Computer Science from the University of Central Florida; he is currently pursuing a Master of Science in Computer Science from UCF. His research interests are in the areas of simulation, graph theory, and computational geometry.

Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He is currently managing Plowshares, an emergency-management simulation project. Previously, he led IST's Computer Generated Forces research projects. Mr. Petty received a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from California State University. He is currently a Ph.D student in Computer Science at UCF. His research interests are in the areas of simulation and artificial intelligence.

Disaggregation Overload and Spreading Disaggregation in Constructive+Virtual Linkages

Mikel D. Petty and Robert W. Franceschini
Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826-0544 USA
mpetty@ist.ucf.edu rfrances@ist.ucf.edu

1. Abstract

A number of projects have successfully linked constructive and virtual simulations. In these linked systems the representation of a military unit can transition from an aggregate unit in the constructive simulation to a set of virtual entities in the virtual simulation. That representational transition, which typically occurs in response to scenario events, is referred to as disaggregation.

Too many disaggregations can overload the linked system with an unsupportable number of virtual simulation entities. One possible cause of such disaggregation overload is spreading disaggregation. Spreading disaggregation occurs when one disaggregation triggers another in a forced sequence.

Spreading disaggregation can be avoided by imposing scenario limits; these may reduce the usefulness of the linked system. It can also be prevented by allowing interaction between the constructive and virtual systems. The latter solution is fraught with implementation challenges and fidelity concerns.

2. Introduction

This introductory section defines the basic terms and concepts of constructive and virtual simulations and constructive+virtual linkages. See (Franceschini, 1995) for a in-depth tutorial on these topics.

2.1 Constructive and virtual simulations

Constructive simulations represent military units (such as companies or battalions) as aggregates; the individual entities (such as tanks or infantrymen) within the units are not explicitly simulated. The location, direction and speed of movement, status, size, and composition of an aggregate unit are maintained in a constructive simulation for the unit as a whole, and are often computed as the result of statistical analysis of the unit's actions. For example, a combat encounter between two opposing battalions may be resolved with a mathematical model of combat that considers the overall combat power of the units rather than representing the individual vehicles and direct fire events than might constitute such an engagement. The military simulations CBS, BBS, and Eagle are examples of constructive simulations.

Virtual simulations represent each individual combat entity as a distinct simulation entity. State information is maintained as needed by the simulation for each simulation entity, each entity is capable of independent action, and combat is resolved at the entity level. SIMNET and DIS are examples of virtual simulation. A virtual simulation's entities may be controlled by either crewed simulators or computer generated forces (CGF).

Table 1 summarizes three important ways in which constructive and virtual simulations differ, though there are exceptions to each entry. Those differences are explained in more detail in (Franceschini,1995).

2.2 Constructive+virtual linkages

A *constructive+virtual* system (or linkage) is a simulation system that includes both a constructive simulation and a virtual system linked together. The systems are linked in such a way that events in one simulation influence or effect events in the other. Furthermore, units and entities may be represented in some fashion in either simulation.

In a typical constructive+virtual scenario, aggregate units are represented in the constructive simulation, where they move and engage in combat. They are located on a terrain database within the constructive simulation. Linked to the constructive simulation is a virtual simulation. It has a terrain database corresponding to all or part of the constructive terrain database.

Characteristic	Constructive	Virtual
Time	Time-stepped Simulation events organized into uniform time increments, computed as quickly as possible.	Real-Time Simulation events transpire continuously, at a rate that ostensibly matches their real-world counterparts.
Terrain representation	Abstracted regular grid of squares or hexagons, assigned abstract terrain types.	Detailed Polygonal or elevation-post based terrain skin with individual features.
Traditional uses	Analysis Simulation results used to evaluate weapons or doctrine.	Training Simulation participation used to train participants.

Table 1 Differences between constructive and virtual simulation.

At some point one or more aggregate units move into the terrain area that has a corresponding representation in the virtual simulation. That fact, and possibly other criteria to be discussed below, trigger the disaggregation of the constructive unit. Control of the constructive unit is transferred from the constructive simulation to the virtual simulation, and the unit is instantiated as individual entities in the virtual simulation. Those individual entities move, engage in combat, and perform other activities that are possible there. Eventually the entities leave the virtual terrain area, or some other criteria is met, and the unit is aggregated. The individual entities are removed from the virtual simulation and the constructive simulation retakes control of the aggregate unit, whose composition is appropriately modified to reflect the virtual events.

Implementing such a linkage is a complex matter. See (Franceschini,1995) for a tutorial on constructive+virtual linkages.

2.3 Benefits of constructive+virtual linkages

Constructive and virtual systems are linked for the following reasons:

1. Simulation analysts can obtain detailed entity level performance and event information for use in constructive simulations. (Franceschini,1994)

2. Trainers can conduct small unit training exercises in virtual battles that are set in the context of larger battles executing in constructive simulation, thereby adding realism and motivation to the training. (Franceschini,1994)

3. The role of constructive simulations in training higher level commanders and the staffs can be enriched by supplementing constructive simulation's aggregate statistical interactions with virtual simulation's detailed entity interactions. (Franceschini,1994) (Downes-Martin,1991)

4. Large numbers of geographically distributed virtual entities whose locations and actions are derived from an overall constructive simulation can provide realistic input to virtual entities that are moving quickly or possess long-range sensors (e.g. a JSTARS platform). (Calder,1994) (Root,1994)

2.4 Some existing constructive+virtual linkages

Several constructive+virtual linkages have been implemented. Table 2 provides a partial list of those systems and references for additional information on them A survey of constructive+virtual linkages can be found in (Kraus, 1995).

3. Disaggregation

This paper is concerned with the disaggregation process, the conditions that might trigger it, and problems that might arise. In this section we shall examine disaggregation in more detail.

3.1 Disaggregation process and triggers

As the constructive units move, they may move into the virtual terrain area. Constructive units in the virtual terrain area may be eligible for disaggregation, i.e. instantiation as virtual entities. Their location may be enough to trigger disaggregation or additional conditions may also be used. Explained below are four disaggregation criteria.

Project Name	Constructive Simulation	Virtual Simulation	Research Agencies	References
Integrated Eagle/BDS-D	Eagle	BDS-D DIS	TRADOC Analysis Command Institute for Simulation and Training U.S. Army STRICOM	(Franceschini, 1992) (Franceschini, 1994a) (Franceschini, 1995b) (Karr, 1992) (Karr, 1993) (Karr, 1994a) (Karr, 1994b) (Powell, 1993) (Root, 1994)
Corps Level CGF	Eagle	ModSAF	U.S. Army Topographic Engineering Center Science Applications International Corporation Raytheon Systems Development Company U.S. Army STRICOM	(Calder, 1994) (Raytheon, 1994a) (Raytheon, 1994b) (Raytheon, 1994c)
BBS/SIMNET	BBS	SIMNET	Advanced Research Projects Agency Naval Research and Development ETA Technologies	(Hardy, 1994)

Table 2. Existing constructive+virtual systems

1. *Location in the virtual terrain area.* The aggregate unit is located within a portion of the virtual terrain area designated in advance as a disaggregation area (perhaps centered on militarily important terrain feature). In general, there may be multiple disaggregation areas specified within the virtual simulation area. When the center of mass of the disaggregated unit (as determined from the entities in the unit) moves out of the disaggregation area, the unit is aggregated. In order to prevent anomalies, one can define the aggregation area to be slightly larger than the disaggregation area; this will prevent a unit on the border of a disaggregation area from being disaggregated and then aggregated repeatedly as it moves along the boundary.

2. *Range to enemy.* The aggregate unit is within a critical range of a disaggregated enemy unit (equivalently, within range of an enemy virtual vehicle). The range may be the maximum detection range of any of the unit's entities' sensor systems.

3. *Intent to interact.* The aggregate unit intends to interact (e.g., employ sensors or conduct direct fire) with some other disaggregated unit (virtual vehicles). The intent would be recognized as a consequence of an action the unit is taking in the constructive simulation.

4. *Operator selection.* A human operator may trigger aggregation or disaggregation using a command interface. This approach allows human intelligence to decide when control transfer is appropriate, so anomalous behavior should not result. This approach is useful as a starting point for building constructive+virtual systems and to permit explicit operator control in order to meet specific exercise objectives. However, automatic disaggregation triggers are often desirable.

5. *Commander's view.* A commander may wish to view different parts of the battlefield at different levels of granularity. As the commander's view shifts around the battlefield, units may be disaggregated or aggregated (Downes-Martin,1991).

Note that the first three criteria are automatic, in that they can be recognized and responded to by the simulation system, whereas the last two are manual, in that they are triggered by human intervention. Note also that the second criteria, *Range to Enemy*, can be seen as a special case of the third, *Intent to*

System Component	Capacity limit based on	Relationship to entity count
Network	Bandwidth, maximum PDU traffic	Each entity produces a required number of PDSs (e.g., Entity State) to be carried on the network.
CGF Simulator	Processor speed and memory	Each entity creates a processing burden on the CGF Simulator, e.g., Remote Entity Approximation, intervisibility, and behavioral control (for internal entities).
CGF Operator	Human span of control	Each entity can potentially be reacted to or controlled by the operator.
Crewed Simulator	Processor speed and memory	Each entity creates a processing burden on the Crewed Simulator, e.g., Remote Entity Approximation and image generation.

Table 3. Relationship of virtual simulation component capacity limits and entity count

Interact, if the reason that range is important is that certain interactions are possible below a given range.

When the disaggregation criteria in effect in the scenario are met, the constructive unit is disaggregated into its component entities. The constructive simulation communicates the location, composition, movement speed and direction, and operational activity to the node of the virtual simulation charged with controlling disaggregated entities; that node is usually a CGF system. The component entities are instantiated by the CGF system as entities in the virtual simulation under its control and the constructive simulation relinquishes control of the unit. The virtual entities are placed in locations that are consistent with the operational activity of the aggregate unit (e.g., if the unit is conducting an assault, the entities will be in an attack formation) and in locations that do not violate realism (i.e., they shouldn't be placed in the middle of a lake). The constructive simulation may maintain a "shadow" or "ghost" unit for the disaggregated unit, which is used to store information about changes in the unit's status as it interacts in the virtual world. ((Franceschini,1992) and (Clark,1994) address issues specific to placement of virtual entities during disaggregation.)

The disaggregated entities conduct direct fire combat (and other interactions) in the virtual simulation under the control of the CGF system. The virtual entities they engage may be disaggregated from other constructive units; in some constructive+virtual systems they may also interact with manned simulators or virtual entities that do not belong to any constructive unit.

Interactions between the constructive and virtual simulations must be mediated by the linkage. An important example of such a connection is indirect fire. In some systems, indirect fire may be conducted between the virtual and constructive simulations, with constructive batteries firing into the virtual world and vice versa.

3.2 Disaggregation overload

Virtual networked simulation systems such as DIS have capacity limits. The different components of the system each have a capacity limit based on some performance characteristic of that component. Though the performance characteristics which impose the limits can vary for different component types, it is often the case that the capacity limit for a system component can be expressed in terms of simulation entity count (the number of entities present in the simulation). Table 3 lists primary components of a DIS simulation system, what their capacity limits are based upon, and how that limit relates to entity count.

Note that the CGF components in Table 3 (CGF Simulator and CGF Operator) are especially important, in that CGF systems are generally used to control the virtual entities that result from a disaggregation.

If the simulation system is limited in the number of simulation entities it can support, then exceeding that limit will lead to problems, including unrealistic results and system failure. In a linked constructive+virtual system, one way to create a situation where the entity capacity of the virtual system is exceeded is to disaggregate too many units

from the constructive simulation. We refer to such a situation as *disaggregation overload.*

Disaggregation overload can be produced by any of the disaggregation criteria listed previously. Manual disaggregation criteria can produce disaggregation overload if the operator selects too many units for disaggregation or if the commander wants to see too much of the forces in too much detail. However, the automatic disaggregation criteria, where disaggregations are triggered by the system automatically in response to simulation events, create a more worrisome risk of disaggregation overload if those events occur too often or without intervening aggregations. In particular, the remainder of this paper will examine one type of disaggregation overload that can result from automatic disaggregations in response to the *Range to Enemy* and *Intent to Interact* criteria; that type is *spreading disaggregation.*

3.3 Spreading disaggregation

We will define spreading disaggregation by example. Suppose a constructive+virtual system is executing a scenario. We will refer to the constructive simulation as C, the virtual simulation as V, and the linked system as C+V. For this example, C+V is using a set of disaggregation trigger criteria that include one based on *Range to Enemy*; specifically, C+V will disaggregate a constructive unit that is within direct fire range of a virtual entity. This criteria is used because in C+V, direct fire combat is possible within constructive simulation C between constructive units and within virtual simulation V between virtual entities, but no direct fire is possible between units in C and entities in V.

Figure 1 shows the example scenario. It includes Blue companies Blue-1 and Blue-2 and Red company Red-3; all three units are within the virtual terrain area (i.e., the portion of the terrain area for which both constructive and virtual representations exist). In Figure 1a, Blue-2 and Red-3 are constructive units; they are engaging in direct fire within the constructive simulation C. However, Blue-1 has already been disaggregated; hence it is a set of virtual entities in V corresponding to a company sized unit. The entities of Blue-1 are moving towards Red-3. As Blue-1 moves toward Red-3 it comes within the disaggregation trigger range of Red-3. The constructive unit Red-3 is now within range of a disaggregated unit and must disaggregate (Figure 1b). Once Red-3 has been disaggregated, it becomes a disaggregated unit within the disaggregation triggering range of Blue-2. That event triggers Blue-2's disaggregation (Figure 1c).

In this way a chain of disaggregations can spread across the battlefield; the phenomenon is called *spreading disaggregation.* Spreading disaggregation could easily and unpredictably result in constructive+virtual exercises where large numbers of units are in short order disaggregated into their component vehicles, leading to disaggregation overload.

The *Intent to Interact* disaggregation criterion can also initiate spreading disaggregation, and might do so at greater ranges than direct fire, depending of the type of interaction. Remote sensing by a surveillance aircraft, jamming by an electronic warfare unit, and indirect fire all occur at longer ranges than direct fire and are interactions that could trigger disaggregation if one of the interacting units is disaggregated. The greater range of these interactions increases the chance of a chain of spreading disaggregations.

4. Some solutions to spreading disaggregation

This section will discuss some solutions to the problem of spreading disaggregation.

4.1 Avoidance

The problem of spreading disaggregation in particular (and disaggregation overload in general) can be "solved" by rendering it impossible or unlikely through scenario design. The simplest way to do so is to design scenarios with few enough constructive units that even if all of the units were disaggregated the total number of resulting virtual entities would not exceed the entity count capacity of the virtual system. This method, while inarguably a solution to the spreading disaggregation problem, eliminates the possibility of executing scenarios larger than the capacity of the virtual system alone, which is one of the main objectives of linking constructive and virtual simulation.

A more subtle form of avoidance is to design scenarios wherein the aggregate units that are likely to be disaggregated are sufficiently dispersed geographically so as to make spreading disaggregation extremely unlikely. This method, a marginal improvement over the simpler form of avoidance, has been used extensively in early demonstrations of constructive+virtual systems.

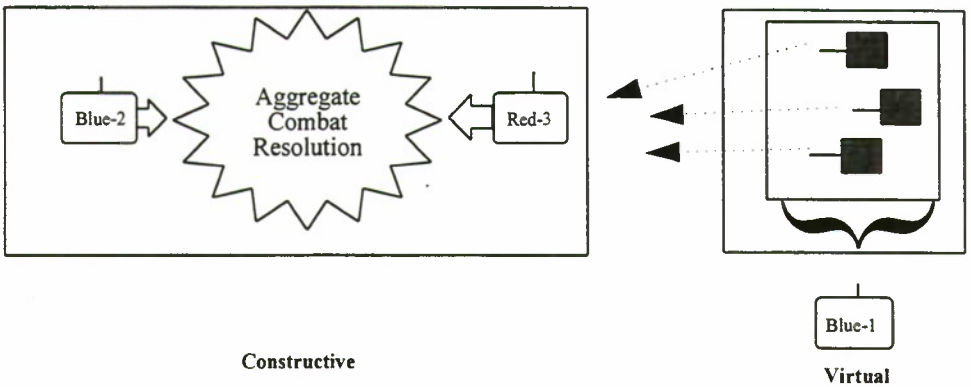


Figure 1a

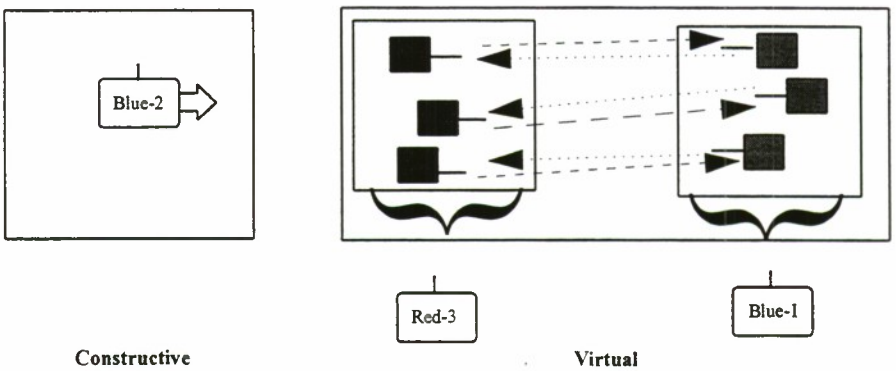


Figure 1b

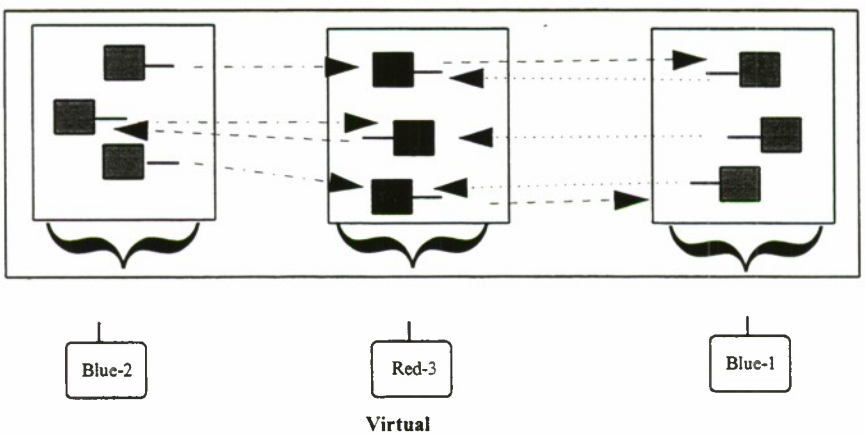


Figure 1c

Figure 1 - Spreading Disaggregation

As an illustration of this idea, consider the scenario used by the Integrated Eagle/BDS-D system at the DIS Interoperability Demonstration during the 16th Interservice/Industry Training Systems and Education Conference. (The Integrated Eagle/BDS-D system uses the *Location in the virtual terrain area* disaggregation criterion for automatic disaggregations and also allows operator initiated disaggregations; therefore, it is not as susceptible to spreading disaggregation as systems which use the other automatic disaggregation triggering mechanisms. However, the scenario used by the system provides a good example of the care that must be taken in developing scenarios for systems that are more susceptible to spreading disaggregation. Such care was taken in constructing the I/ITSEC scenario so that the basic concepts of a constructive+virtual linkage could be demonstrated without complicating the battlefield.) This scenario was constructed so that it was known in advance which units would be disaggregated, which units would engage each other, etc. Units which were not selected for disaggregation were, in general, not close enough to the disaggregated units to cause a need for interactions between constructive units and virtual entities (with the exception of indirect fire volleys from the constructive simulation to the virtual simulation).

4.2 Interaction between simulations

Spreading disaggregation can occur because the presence of one disaggregated unit triggers the disaggregation of the next one. The successive disaggregations are forced because the interactions that may occur between the units can be resolved only constructive↔constructive or virtual↔virtual. It is possible to eliminate the forcing of disaggregations by permitting interactions between the constructive and virtual simulations, i.e., constructive↔virtual. Given such a capability, the constructive+virtual system could choose not to disaggregate if spreading disaggregation (or some other type of disaggregation overload) would result. This solution was first suggested in (Trinker,1994).

In the example given of spreading disaggregation, a *Range to Enemy* disaggregation criterion leads to a chain of disaggregations because satisfying the criterion signals the possibility of direct fire. In existing constructive+virtual systems, direct fire may only be performed within the constructive and virtual components. Providing the capability of constructive↔virtual direct fire would halt spreading disaggregation because the presence of a disaggregated unit within direct fire range is no longer an automatic trigger for disaggregation. Spreading disaggregation would have been prevented in the example by constructive↔virtual direct fire between Blue-1 and Red-3 or between Red-3 and Blue-2.

However, experiences with allowing indirect fire between the constructive and virtual components of a C+V simulation have shown the difficulties that can be encountered in implementing cross-simulation interaction. In the Integrated Eagle/BDS-D system, indirect fire can be conducted in both directions between aggregate units in Eagle (the constructive simulation) and virtual entities in DIS (the virtual simulation). Some of the problems encountered in implementing that interaction include timing of the indirect fire rounds and positioning of their detonations. For example, when indirect fire is sent from Eagle into DIS, an volley representing the total number of rounds fired during Eagle's current five minute timestep is shipped to DIS in one instant. The constructive to virtual interface is responsible for parceling out the fire over the correct amount of real time in the virtual simulation. As another example, indirect fire rounds fired from DIS into Eagle must be gathered and "aggregated" (in both time and space) into one volley in order for Eagle to correctly assess damage to its units. More information about the implementation of indirect fire across the constructive and virtual boundary in the Integrated Eagle/BDS-D system can be found in (Karr, 1993) and (Karr, 1994a).

Direct fire would be even more problematic. One obvious reason is the importance of intervisibility in conducting direct fire. How can intervisibility be determined between virtual entities, which have specific locations in the virtual terrain database, and aggregate units consisting of a number of abstract entities with no specific location? How can direct fire be realistically conducted without intervisibility? Difficult enough if the virtual entities are controlled by a CGF system, these questions become vastly more troublesome when the virtual entities involved include crewed simulators. How can the crews shoot at the entities of aggregate units when those entities cannot be seen? How can they take cover from those units' return fire?

There are additional problems to consider in implementing constructive↔virtual direct fire. Several are identified and considered in (Trinker,1994).

We list here the problems identified by (Trinker, 1994); readers interested in proposed solutions to these problems should refer to that paper. Normally, direct fire is partitioned into two separate events: hit assessment and damage assessment; each of these events raises implementation concerns for constructive↔virtual direct fire. For hit assessment, how are specific entities or units identified as being hit? How many entities are hit? When are the entities hit? How is damage assessed, and which simulation assesses it?

Generalizing from direct fire to a wider range of interactions (and from *Range to Enemy* to *Intent to Interact* disaggregation criteria), the constructive↔virtual interaction solution becomes more challenging. If automatic disaggregations forced by *Intent to Interact* are to be avoided, then constructive↔virtual interaction must be possible for each of the types of interactions considered by the *Intent to Interact* criterion. If *Intent to Interact* considers sensing, then constructive↔virtual sensing must be implemented; if *Intent to Interact* considers electronic warfare, then constructive↔virtual electronic warfare must be possible, and so on.

5. Conclusions

Spreading disaggregation (and disaggregation overload) is a difficult problem that limits the utility of constructive+virtual linkages. The best solution currently known to the problem of spreading disaggregation is implementing constructive↔virtual interactions. Unfortunately, implementing those interactions would entail significant effort and would face serious realism concerns.

6. Acknowledgments

This research is sponsored by the U. S. Army Simulation, Training, and Instrumentation Command (STRICOM) as part of the Signal Intelligence/Electronic Warfare project (contract N61339-93-C-0091). IST's ongoing work in constructive+virtual simulation is sponsored by STRICOM and the U. S. Army TRADOC Analysis Center (TRAC) as part of the Integrated Eagle/BDS-D project (contract N61339-92-K-0002). That support is gratefully acknowledged.

7. References

Calder, R. B. and Evans, A. B. (1994). "Construction of a Corps Level CGF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 4-6 1994, Orlando FL, pp. 487-496.

Clark, K. J. and Brewer, D. (1994). "Bridging the Gap Between Aggregate Level and Object Level Exercises", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 4-6 1994, Orlando FL, pp. 437-442.

Downes-Martin, S. (1991). "Vehicle Level Wargaming for Senior Commanders: Integrating Wargames with Vehicle Level Simulations", *Unpublished*, February 1991, 33 pages.

Franceschini, R. W. (1992). "Intelligent Placement of Disaggregated Entities", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 20-27.

Franceschini, R. W. and Karr, C. R. (1994). "Integrated Eagle/BDS-D: Results and Current Work", *Proceedings of the Eleventh Workshop on the Standards for the Interoperability of Defense Simulations*, September 26-30 1994, Orlando FL.

Franceschini, R. W. and Petty, M. D. (1995a). "Linking constructive and virtual simulations in DIS", *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing & Control and Dual-Use Photonics*, Orlando FL, April 17-21 1995.

Franceschini, R. W. (1995b). "Integrated Eagle/BDS-D: A Status Report", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation,* Institute for Simulation and Training, May 9-11 1995, Orlando, FL.

Hardy, D., Healy, M. (1994). "Constructive and Virtual Interoperation: A Technical Challenge". *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 4-6 1994, Orlando FL, pp. 503-507.

Karr, C. R., Franceschini, R. W., Perumalla, K. R. S., and Petty, M. D. (1992). "Integrating Battlefield Simulations of Different Granularity", *Proceedings of the Southeastern Simulation Conference 1992*, October 22-23, Pensacola FL, pp. 48-55.

Karr, C. R., Franceschini, R. W., Perumalla, K. R. S., and Petty, M. D. (1993). "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, March 17-19 1993, Orlando FL, pp. 231-239.

Karr, C. R. and Root, E. D. (1994a). "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 4-6 1994, Orlando FL, pp. 425-435.

Karr, C. R. and Franceschini, R. W. (1994b). "Status Report on the Integrated Eagle/BDS-D Project", *Proceedings of the 1994 Winter Simulation Conference*, Society for Computer Simulation, Orlando FL, December 11-14 1994, pp. 762-769.

Kraus, M. K., Stober, D. R., Foss, W. F., Franceschini, R. W., and Petty, M. D. (1995). "Survey OF Constructive+Virtual Linkages", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 9-11, Orlando, FL.

Powell, D. R. and Hutchinson, J. L. (1993). "Eagle II: A Prototype for Mult-Resolution Combat Modeling", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, March 17-19 1993, Orlando FL, pp. 221-230.

Raytheon (1994a). "System Specification for the Corps Level Computer Generated Forces", *Contract No. DACA76-93-D-0007 CDRL Sequence No. A0010*, Raytheon System Development Company, July 22 1994.

Raytheon (1994b). "Draft System Segment Design Document for the Corps Level Computer Generated Forces", *Contract No. DACA76-93-D-0007 CDRL Sequence No. A0011*, Raytheon System Development Company, August 8 1994.

Raytheon (1994c). "Implementation Plan for the Corps Level Computer Generated Forces", *Contract No. DACA76-93-D-0007 CDRL Sequence No. A0012*, Raytheon System Development Company, August 8 1994.

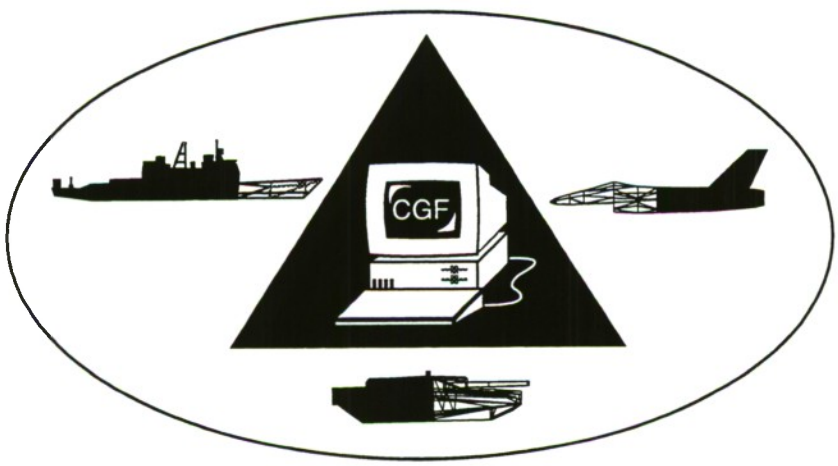
Root, E. D. and Karr, C. R. (1994). "Displaying Aggregate Units in a Virtual Environment", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, May 4-6 1994, Orlando FL, pp. 497-502.

Trinker, A. (1994). "General Architecture for Interfacing Virtual and Constructive Simulations in DIS Environment", *Technical Report IST-TR-94-28*, Institute for Simulation and Training, September 14 1994.

8. Authors' Biographies

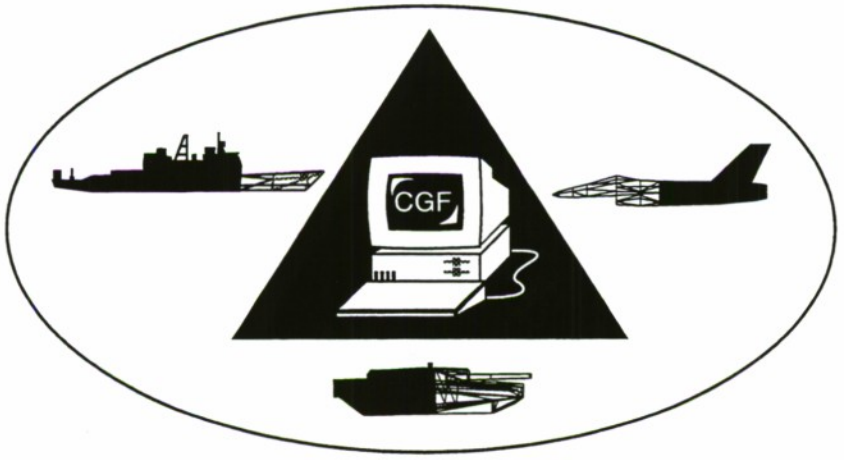
Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He is currently managing Plowshares, an emergency management simulation project. Previously he led IST's Computer Generated Forces research projects. Mr. Petty received a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from California State University, Sacramento. He is a Ph.D. student in Computer Science at UCF. His research interests are in simulation and artificial intelligence.

Robert W. Franceschini is a Principal Investigator at the Institute for Simulation and Training. He leads the Integrated Eagle/BDS-D project, which was the first research effort to successfully link a constructive simulation with a virtual simulation. Mr. Franceschini earned a B.S. in Computer Science from the University of Central Florida; he is currently pursuing an M.S. in Computer Science at UCF. His research interests are in simulation, graph theory, and computational geometry.


CGF

Session 3b: Reasoning II

Lehman, Carnegie Mellon University
Tambe, ISI, USC
Hieb, George Mason University


CGF

Natural Language Processing for IFORs: Comprehension and Generation in the Air Combat Domain

Jill Fain Lehman, Julie Van Dyke, and Robert Rubinoff
Carnegie Mellon University
Pittsburgh, PA 15213
jef@cs.cmu.edu

Abstract

In support of the Soar/IFOR project's goal of providing intelligent forces for distributed interactive simulation environments [Laird *et al.*, 1995], the NL-Soar project works toward the implementation of a full natural language capability for Air-IFOR agents. In this paper we discuss the design of that language capability (NL-Soar) and its integration into TacAir-Soar agents. In particular, we demonstrate how NL-Soar's linear complexity, interruptibility, and atomaticity of language processing provide language comprehension and generation processes that do not compromise agent reactivity.

1 Introduction

Autonomous intelligent forces (IFORs) play an increasingly critical role in both large-scale distributed simulations and small-scale, focused training exercises. An IFOR is a complex agent that requires diverse capabilities to perform at a useful level of functionality. Since an IFOR's role will often be to replace one or more individuals in an engagement, the ability to communicate in natural language can be a key capability contributing to its overall performance. An agent that is rigid in its communicative ability may introduce a brittleness into the simulation (i.e. a tendency to fail in unexpected ways) that has nothing to do with imperfections in strategic or tactical knowledge. Thus, in building TacAir-Soar agents to participate in beyond-visual-range combat [Laird *et al.*, 1995], an NL capability is needed to ensure reactive, human-like performance in basic interactions among pilot, wing, and air intercept control (AIC).

In [Rubinoff and Lehman, 1994a] we identified three main characteristics of communication during air combat that present challenging areas of research: (1) it occurs in real-time, (2) it must seamlessly integrate with the agent's non-linguistic capabilities, e.g. perception, planning, reasoning about the task, and (3) its content must be comprehended and generated in accordance with performance data, i.e. with all of the idiosyncratic constructions, ungrammaticalities, and self-corrections found in real language. Within the context of these research issues, we introduced NL-Soar, a language comprehension and generation capability designed to provide integrated, real-time natural language processing for systems built within the Soar architecture [Lewis, 1993; Nelson *et al.*, 1994a; Nelson *et al.*, 1994b; Rubinoff and Lehman, 1994b]. In this paper we concentrate on issues (1) and (2), exploring our progress toward their solution using NL-Soar in Soar-based Air-IFOR agents.

2 Demands of reactivity

The naive approach to communication between agents, and the one available using off-the-shelf technology, treats language as front-end and back-end interfaces. Messages are comprehended by a front-end module, which creates a system-dependent representation of the message that can be used by the other modules responsible for the agent's behavior. Similarly, when an agent needs to send a message, that same representation is passed to a back-end module that generates an output message to be directed to other agents.[1]

This makes language an all-or-nothing endeavor, the implications of which can be seen in Figure 1. In this typical tactical air scenario, blue is flying an intercept (1) and is actively pursuing the goal of achieving its launch acceptability region (LAR) when an incoming message arrives (2). The message is buffered until the current goal is achieved and blue has fired a missile (3). Next, processing of the input begins (4); it ends sometime after red has returned fire (5) and (6). Only after the communiation has been understood can blue begin its evasive maneuver (7).

It is clear that reactivity is compromised if understanding must be postponed until the current

[1]The approach being described here does not depend in any way on the content of the message or the style of language accepted and generated. Thus it would apply equally whether the language passed is natural language or a formal communication protocol (such as CCSIL [Salisbury, 1995]).

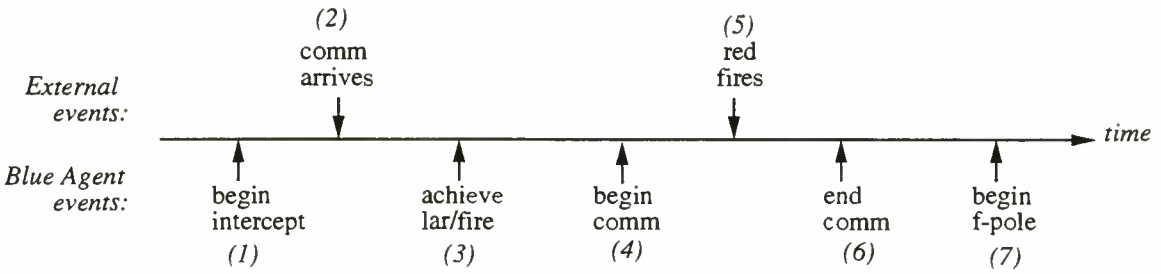


Figure 1: All-or-nothing: a communication model that compromises reactivity

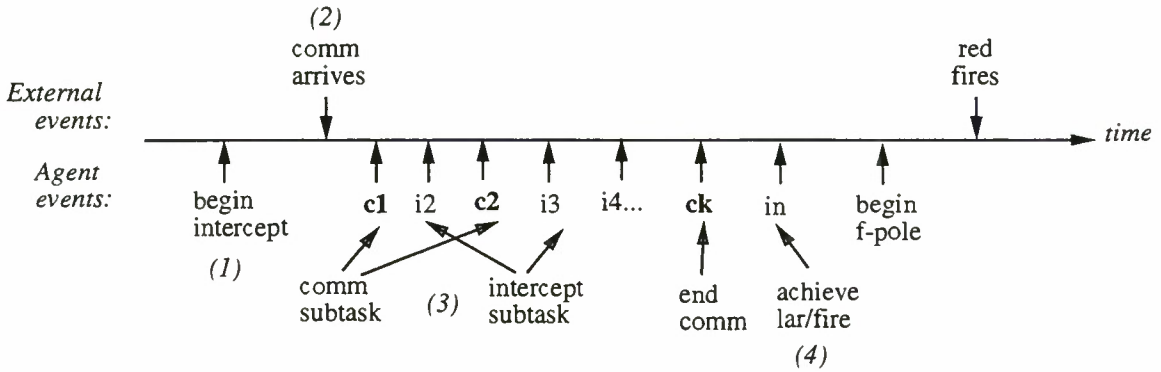


Figure 2: Reactive communication: interleaving comm and non-comm subtasks

goal has been accomplished, and then is pursued to the exclusion of all else. In particular, two cases are cause for concern. Consider first what happens at (2) if the content of the message is relevant to the situation at the time it is received. In this case, buffering the message leads, at best, to wasted processing in the future (when the message has become obsolete). At worst buffering compromises the decision making of the agent by precluding access to timely, necessary information. To remove this possibility, we could modify the control of the agent to always attend to communication needs first. But this would simply put us in the second problematic situation more often.

In this second case (4), if the content of the message is not critical, devoting processing to it rather than other things can compromise the agent's reactivity as well. In short, shutting out either communication processes or non-communication processes can be equally dangerous. The point, of course, is that you can't tell which situation you will be in until you process the message, at which time it is too late to change your mind.[2]

Figure 2 gives a more desirable version of the same task events. Again, the pilot is flying an intercept (1), trying to achieve firing position when a message arrives (2). The message is attended to immediately, its processing interleaved with the ongoing effort to achieve LAR (3). In this example, the message is completely processed by the time the pilot is in a position to fire (4), and evasive maneuvers can be started immediately, well before red returns fire.

The model in Figure 2 overcomes the problems in the simpler model of Figure 1 by intertwining the different strands of agent behavior at the sub-

[2]Dedicating a separate, parallel process to communication might ameliorate the problem but won't necessarily solve it. A separate process will be able to comprehend or generate the message while the agent is performing other tasks, but will have to work in isolation, i.e. cut off from the changing situation and goals of the agent. To the extent that there is relevant information that is unavailable during communication processing, the agent may formulate interpretations or communications that are inappropriate or out of sync. To the extent that the relevant information is communicated to the language process, parallelism is lost. In the tactical air domain information is updated quickly, and so an increasing proportion of CPU cycles will be necessary to keep the two processes in sync. Thus, to maximize reactivity, we conjecture that a separate process for communication would be more costly and no more effective than the method outlined in the following section.

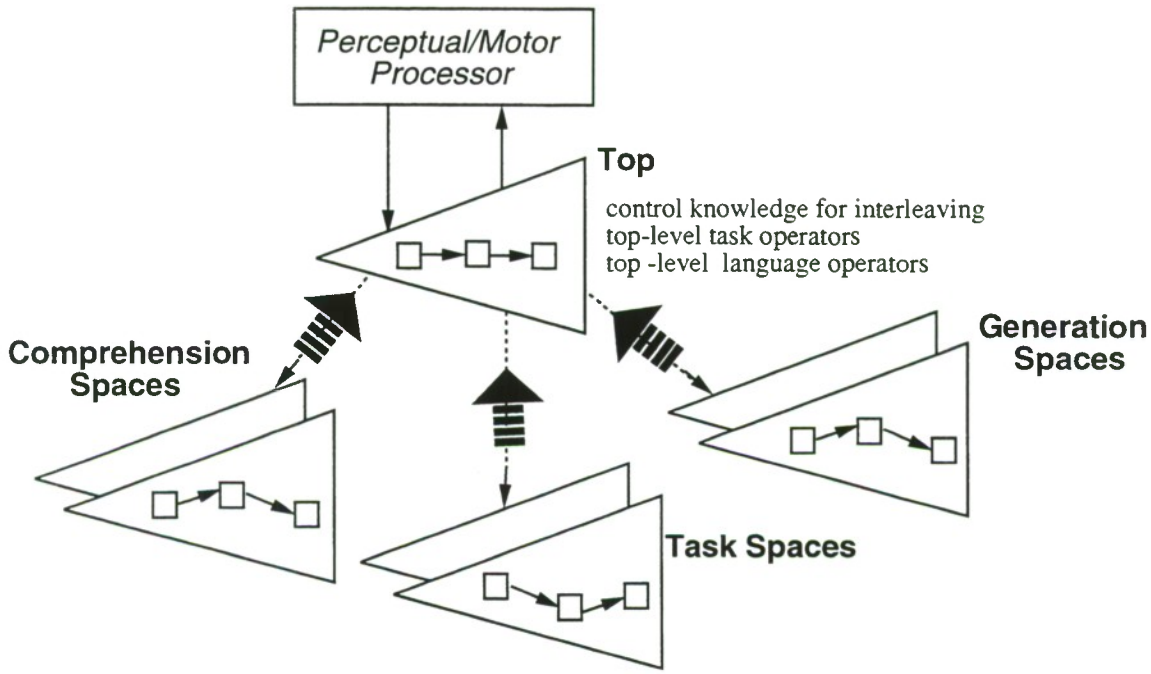


Figure 3: An Example of the Top-state Control Model

task level rather than at the full task level. In other words, we can view the all-or-nothing model as a degenerate case of Figure 2, one in which the granularity of the interleavable components is as large as possible. As we have seen, the disadvantage of choosing the maximal grain size is that the components are too large for the agent to behave in a timely fashion.

3 Achieving Interleavable Communication

For NL-Soar to provide a reactive, interleavable language capability for IFOR agents, the system as a whole must have three properties: *linear complexity*, *interruptability*, and *atomaticity*. The first property, linear complexity, means that processing to understand or generate a message must take time that is roughly linear in the size of the message. This is necessary to keep pace with human rates of language use. The second property, interruptability, ensures that time-critical task behaviors cannot be shut out by language processing (and vice versa). The third property, atomaticity, ensures that if language processing is interrupted, partially constructed representations are left in a consistent and resumable state.

To understand how NL-Soar provides the desired communication model, we must first briefly review the components out of which Soar systems are organized. Figure 3 is a graphical representation of a hypothetical Soar system that uses NL-Soar for comprehension and generation. Linguistic processes, like all processes in Soar, are cast as sequences of *operators* (small arrows) that transform *states* (boxes) until a goal state is achieved. The triangles in the picture represent *problem spaces* which are collections of operators and states.[3] The comprehension problem spaces contain operators that use input from the perceptual system to build syntactic and semantic structures on the state; the generation problem spaces contain operators that use semantic structures to produce syntactic structures and motor output. Note that the problem space labelled Top is the only space connected to the perceptual and motor systems and it is this space that is designated by the Soar architecture; all other problem spaces are provided by the system designer.

The dotted lines in the figure represent Soar impasses which arise automatically when there is a lack of knowledge available in the current problem space. When an impasse arises, processing continues in a subspace until the goal state in the subspace is reached. Note that impasses are a general recursive structure (a subspace can impasse into another subspace) that gives rise to a goal/subgoal hierarchy, or *goal stack*. The thick banded arrow

[3] For more details on how Soar uses problem spaces, states and operators to organize its processing see [Laird *et al.*, 1987; Laird *et al.*, 1995].

that overlays the impasse represents the resolution of the impasse, and the new knowledge (called *chunks*) that results from Soar's learning mechanism. Chunks capture the work done in the subspace, making it available in the superspace without impasse during future processing. This means that when a system structured as in Figure 3 is *fully chunked* all of its behavior will be produced by operators in the Top space.

We now have all the pieces to build an interleavable language capability. In the following sections, we address how to achieve linearity, interruptability, and atomaticity using these components. For the time being we will consider communication only in systems where the desired behavior shown in Figure 2 would occur completely within the Top problem space when fully chunked. We call a system organized in this way, a *Top-state control* model.[4]

3.1 Achieving Linear Complexity

Communication in an IFOR must occur in real-time to keep pace with the flow of human events. This is not a statement about how fast the system must run, per se. Rather, it is a theoretical statement about how processing must occur within the system. Although there is some variability (some words do reliably take longer to process than other words), in general, the amount of time taken by people is linear in the number of words in the utterance. A number of design constraints follow from this simple regularity [Lehman *et al.*, 1996], e.g. construction of the meaning of the sentence must proceed incrementally, and different knowledge sources (syntax, semantics, pragmatics) must be applied in an integrated rather than pipe-lined or multi-pass fashion. NL-Soar provides these properties [Lehman *et al.*, 1991a; Lewis, 1993]. Briefly, the system relies on Soar's notion of impasse to control the search through its linguistic knowledge sources, and then on Soar's learning mechanism to compile the disparate pieces of knowledge into an integrated form that can be applied directly (i.e. in approximately constant time/word) in the future.

Figure 4 depicts the process graphically for one type of language operator, expanding the left portion of Figure 3. Consider the arrival of a new word into the Top state and assume that the system has not encountered the word in a similar context in the past (i.e. the system has no pre-chunked knowledge about how to process this word). Once the word has been attended to, the learn-comprehension operator will be selected, after which an impasse will arise. Problem solving

[4] As we will see in Section 4, this is not the only structure permitted by Soar, but it is a valid organization and the simplest place to begin.

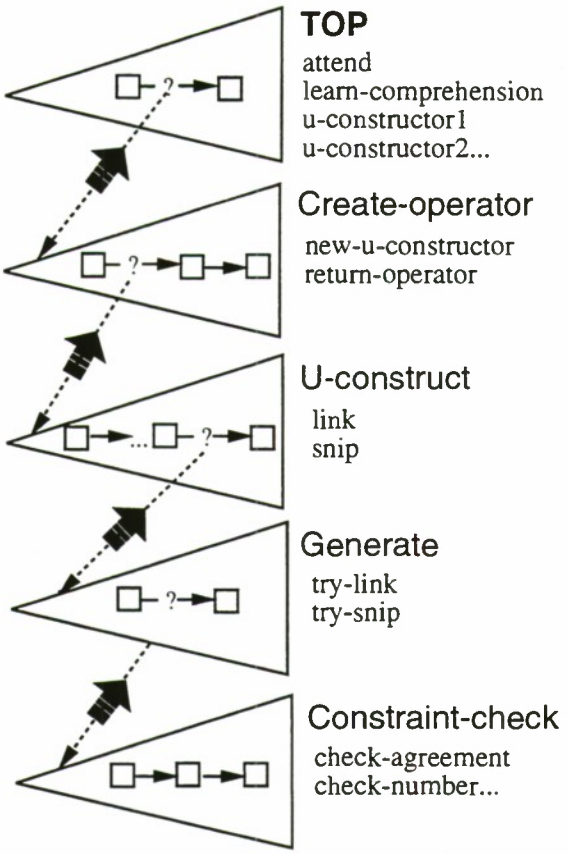


Figure 4: Achieving linearity through learning

will continue in the Create-operator space which will generate a symbol for a new *u-constructor*. A u-constructor is a language operator that fits the new word into the current syntactic structure for the message. The u-constructor is composed piecemeal in the U-construct space which performs links and snips on syntactic trees based on knowledge provided by Generate and Constraint-check. As the goal of each subspace is achieved, each impasse is resolved, creating chunks. Only two kinds of chunks concern us here. The implementation of the u-constructor is contained in the chunks created when the impasse between Create-operator and U-construct is resolved. This means that the syntactic tree that resulted from the sequential links and snips that were done in the lower spaces will now be produced immediately whenever this u-constructor executes. The u-constructor itself is returned from Create-operator to the Top space, resulting in a chunk that tells when this u-constructor can apply in the future. Note that the next time this word is seen in a similar contect, this chunk will propose the new u-constructor directly in the Top state. In other words, *once we have learned the top-level opera-*

tor, no impasse will occur. Instead, the (possibly lengthy) problem solving that took place in the subspaces has been compiled into a single Top-space operator that executes directly to build the relevant syntactic structure on the Top state.

Figure 4 shows how the application of general knowledge about syntax is contextualized and made efficient. A similar story can be told about *s-constructors*, the Top-space operators that fit the new word into the semantic and discourse structures maintained on the Top state. Thus, once behavior is fully chunked, the arrival of a message results in only a small number of Top operators per word, the linear complexity we were after. Equally important, the language process itself is now represented in the Top space in terms of more finely-grained operators (u- and s-constructors) that create the opportunity for interleavability. On the generation side, of course, there is a different task decomposition producing a different set of Top-space operators, but the principle is the same.

3.2 Achieving Interruptability

In Soar, agent behavior is produced by the application of operators to a state. Moreover, the architecture defines the application of an operator as a non-interruptable unit of work. In other words, once an operator has been selected for application, all the state changes associated with that operator are guaranteed to be made before any other operator is selected. What does this mean for NL-Soar? In short, it means that the Top-level language operators dictate the granularity of the interleavable components. To anchor the point in the context of Figure 4, once a u-constructor exists, we cannot interleave changes to the syntax tree with other non-linguistic tasks. Put more strongly, once the u-constructor is selected, all other subtasks are shut out for the duration of its application. In addition, if the Top state changes during the application of the u-constructor (via perception), those changes are effectively invisible until the u-constructor's state changes have been made.[5]

How is this situation different from the one in Figure 1, where lack of interruptibility meant reactivity was diminished to the point of inviting wasted work, if not disaster? The difference here is that the granularity of NL-Soar's operators is small enough to allow interruptibility below the full task level. The current scheme separates the work of attention from work done to the syntactic tree (u-constructors) from work done to the semantic and discourse models (s-constructors). Thus, the current comprehension capability allows for interruption between each set of state changes. Note, however, that we could have made this choice differently. We could, for example, build both syntactic and semantic structures in the impasse under the learn-comprehension operator. The resulting Top-space *comprehension operator* would effectively bundle all of comprehension into a single operator.[6] Alternatively, we could make link and snip the Top operators, giving an even finer grain. Although it is clear that the architecture permits a wide range of choices, choosing the right granularity is not a wholey unprincipled exercise. In general, the more work encompassed by a Top operator, the more specific will be the conditions under which it can apply. The more specific the conditions the less transfer of the knowledge to new situations and the more learning events will be required to get fully chunked language behavior. On the other side, the less work encompassed by a Top operator, the more operators per word there will be, until, eventually, the number will reflect some non-linear quantity (e.g. the size of the parse tree). In Section 4, below, we demonstrate how the operator granularity we have chosen allows both transfer and interleaving while maintaining linearity.

Now that we have language operators of a size that allows interruptibility, the next question that needs to be addressed is: how do you decide which type of operator, linguistic or non-linguistic, to select next? Many control schemes are possible, ranging from random selection to a complete partial ordering over all the operators in the system, to always attending to communication first (or last). In integrating NL-Soar with TacAir-Soar we will use random selection for its simplicity. What is important to remember, however, is that under Top-state control the selection decision is made on an operator by operator basis, not task by task.

3.3 Achieving Atomaticity

Recall that atomaticity ensures that if language processing is interrupted, partially constructed representations are left in a consistent and resumable state. Given our discussion above, it would seem that the architecturally enforced non-interruptability of operators would guarantee atomaticity as well. This is certainly true if all of the language behavior is impasse-free. Suppose, however, that the system is in the middle of learning a new u-constructor or s-constructor, as in Figure 4, when state changes create a preference for a non-linguistic Top-space operator. In this case,

[5] This is an overstatement. In fact, it is possible to encode knowledge in Soar in such a way that it is tied only to the state, not to any particular operator. Such knowledge will lead to state changes regardless of what operator is being applied. Since most task knowledge is tied to task operators, however, the discussion above is still a useful way to think about what's going on.

[6] An early version of NL-Soar did, in fact, use this scheme [Lehman *et al.*, 1991b].

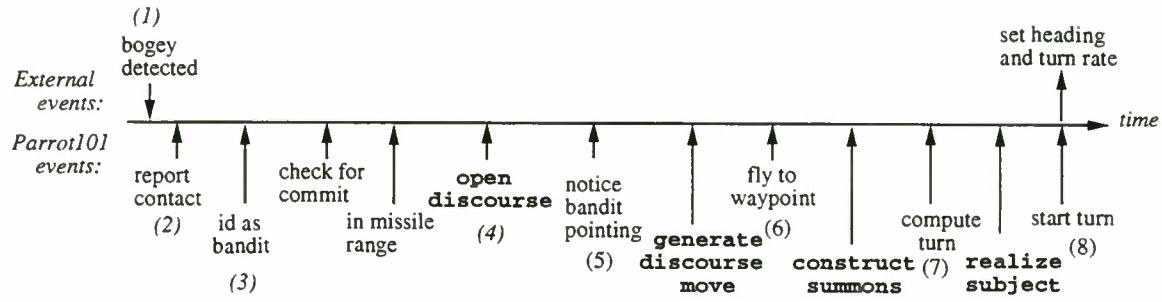


Figure 5: The lead TacAir agent composes a message while tracking a threat and flying

once the operator currently being applied in the lowest subspace is finished, the task operator will be selected in the Top space and the language goal stack will collapse. Can we be sure that we have been left in a consistent state so that language processing can be smoothly resumed?

The answer is yes because the design of NL-soar ensures that no changes are actually made to the language data structures on the Top state until the u-constructor is returned. Look again at Figure 4. The only operator that can result in changes to the Top state is Create-operator's return-operator. But if it is being applied when a preference is created for a Top-space task operator, then we know it will complete, the results will be returned, and the u-constructor proposal chunk will be built. If the subspace operator is not the return-operator, no results will be returned from the top-most impasse and no proposal chunk will be built for the u-constructor. Observe, however, that the conditions that led to the learn-comprehension operator in the Top space may well still obtain. So once the task operator has been applied, language may be resumed. Since no u-constructor was built, the system will have to rebuild the goal stack to continue. In practice, the situation is not as bad as it sounds because chunks may have been built in the subspaces during the previous learn-comprehension processing that were not specific to the particular u-constructor. These chunks will transfer to the current situation and the impasses that created them will be avoided.

4 Bringing it all together in TacAir-Soar

In Section 2 we argued that a communication capability for IFORs had to have three properties: linear complexity, interruptability, and atomaticity. In the previous section we introduced the Top-state control model in which whole tasks are interleaved on an operator-by-operator basis and communication is just another task. One of the interesting characteristics of systems organized as in Figure 3 is that the goal stack is never shared across linguistic and non-linguistic tasks; the need to understand or produce a message pulls the system out of a task goal stack. As a result, Top-state task operators, like the Top-state language operators, tend to represent subtasks of fairly short duration.

In contrast, systems like TacAir-Soar are composed of a Top task operator of very long duration, and a goal stack that reflects many levels of abstraction of that task. Each level stays active as long as it is being carried out. In particular, TacAir uses Soar's Top state to keep track of the "execute-mission" task, which stays active for the entire simulation. Under this will be a stack of sub-tasks, such as "mig-sweep", "intercept", "employ-weapons", and so on, each representing a more detailed view of what the agent is currently trying to do. Much of TacAir's knowledge of its current situation and goals is stored in sub-states associated with these subtasks, not on the Top state.[7] Thus, if TacAir switched to language in its Top state, it would lose much of this knowledge. Clearly, TacAir-Soar is incompatible with the Top-state control model outlined above. To understand how to modify Top-state control without sacrificing linearity, interruptibility and atomaticity, we must answer the question: what role, exactly, does the Top state play in maintaining each property?

For linear complexity, the role played by the Top state is simply a place to apply the so-called Top-state operators. In reality, what is critical for linear complexity is that there is an effective procedure for building the top-level language operators, and that only a small number of them are necessary for each word in the message. For interruptability and atomaticity, the Top state does play a more central role. Specifically, it must be the place where Top-level language operators leave

[7] A fuller description of TacAir-Soar can be found in [Laird *et al.*, 1995].

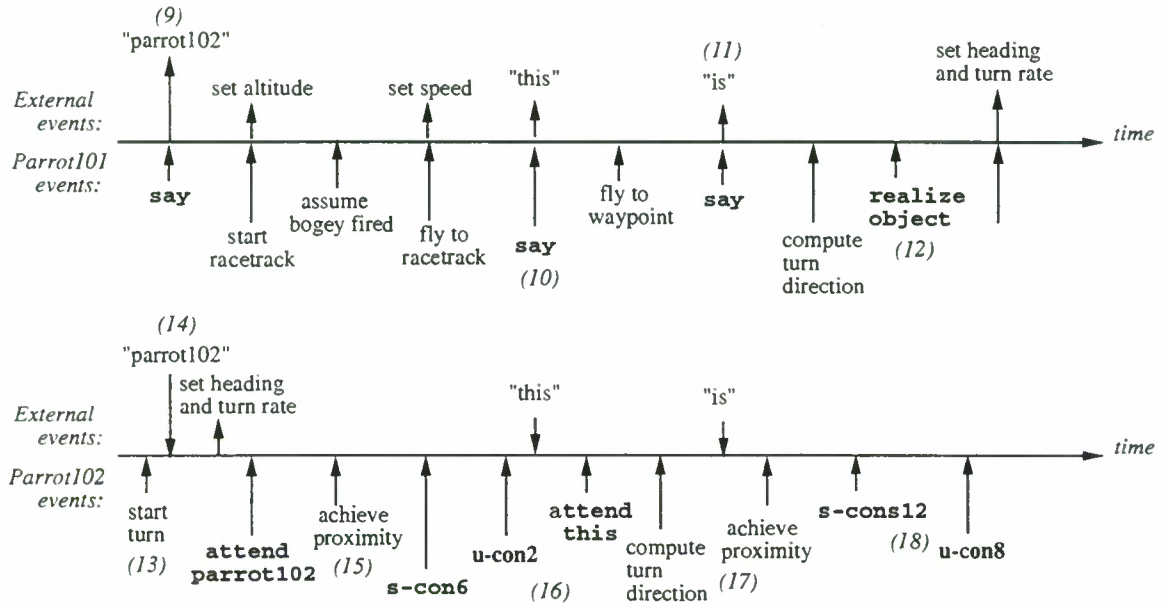


Figure 6: Figure 5 continued: Pilot continues to talk as wing begins to listen

their results because it is the only state that is guaranteed to still be in the goal stack when language processing resumes. Thus, *where* top-level language operators are applied is immaterial as long as they leave their results on the Top state where they can be found whenever, and wherever, language processing resumes.

Separating the question of where top-level language operators are applied from the question of where they leave their results allows us to define a variety of *virtual Top-state control* schemes. The simplest one, and the one we use when integrating NL-Soar with TacAir agents, is to interleave language operators with whatever task operators are available in the lowest problem space in the goal stack. Because the goal stack grows and shrinks over time, the interleaving of communication will take place more or less throughout the range of non-linguistic subtasks. The simplicity of the integration is extended by allowing the architecture to decide randomly between language and non-language operators whenever both types are applicable in the current situation.

Figures 5 through 7 capture a portion of the behavior of two TacAir-Soar agents running with a fully-chunked NL-Soar under virtual Top-state control.[8] In the scenario depicted, two pilots fly F14s as a section with a single red plane flying against them. Parrot101 is the lead and Parrot102 is the wing. The timelines in the figures show the operators that each agent executed in a particular engagement, together with those events in the external world that affect or depend on their behavior. Language operators are indicated via bold-face. For simplicity, the representation does not try to preserve the goal-subgoal relationship of the task operators.

In the time prior to the first event shown in Figure 5, the two planes have begun to fly in a racetrack configuration. The portion of behavior we are interested in begins when the lead notices the bogey (1), and must communicate the relevant information to its wing. The report-contact operator (2) posts a *communicative goal* on the Top state indicating that the agent wants to say something. Interleaving begins (somewhat unevenly due to the random control scheme) at (3). First, three task operators are executed in which the agent determines that the bogey is in fact a bandit, decides to check whether the commit criteria have been satisfied (they have not), and notices that the bandit is within missile range. Then, at (4), language operators begin to compose the message according to communication doctrine. The first step in any lead-wing communication is the

[8] Requiring NL-Soar to "learn while doing" would be equivalent to expecting the pilot to learn the domain language while flying the plane in battle. Consequently, we use off-line training to allow NL-Soar to learn from experience in a non-real-time setting. This gives the system the time it needs to integrate its disparate knowledge sources into the top-level operators discussed in Section 3.1. It is this highly compiled form of language knowledge that models an experienced pilot and provides real-time language behavior on-line.

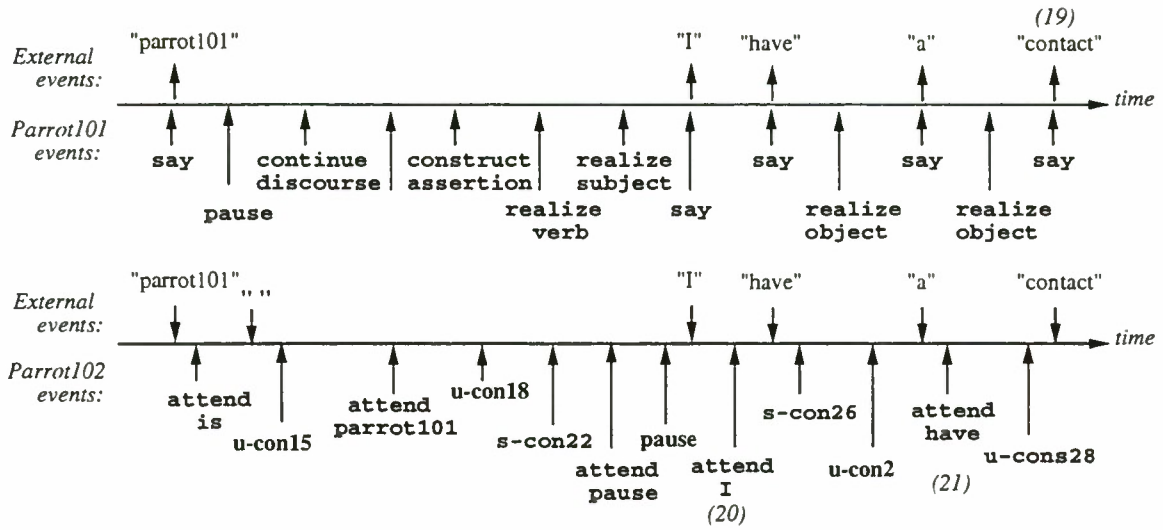


Figure 7: Figure 6 continued: completion of summons generation

exchange of callsigns, here, the sentence *Parrot102 this is Parrot101.* This is a domain-dependent instance of the more general class of utterances we call summons (for example the telephone exchange *John? It's Jill.*) The summons is constructed piece by piece using top-level generation operators (in boldface). Figure 5 shows this linguistic process interleaved with operators that contribute to situation awareness (5) and operators that fly the plane (6), (7), and (8).

Figure 6 continues the timeline for Parrot101 and introduces Parrot102 at the point just before the first word of the summons arrives into the agent's input buffer. The timelines are aligned by the linguistic output of Parrot101 and the linguistic input of Parrot102.

To this point in the scenario, the wing has simply been flying a racetrack with the lead. At (9) Parrot101 outputs the wing's callsign in the upper timeline. Note that this is done even though the construction of the remainder of the summons is still being interleaved with non-linguistic subtasks (10) through (12); both generation and comprehension are incremental. Meanwhile, shortly after Parrot102 has begun to turn (13), the callsign is heard (14). The lower timeline continues with comprehension of the first few words of the summons ((16) and (18)) interleaved with operators that keep the wing in formation ((15) and (17)). Note that the s- and u-constructors for the word *this* (18) fire after the word *is* has already been heard. This is partly because the lead's message is coming out quickly, and partly because the wing's attention has been focused on flying the plane. The input buffer that holds unattended speech has a decay rate; as in people, if speech goes unattended long enough (as it may if the pilot is in a stressful situation), it simply disappears from the buffer.

Figure 7 continues the interchange to the point that Parrot101 outputs the final word of the summons (19). There is no interleaving in this portion of the trace because both pilots are simply flying the long leg of the racetrack where no task operators are proposed. Notice that by the time the lead has begun the second portion of the summons, the wing has caught up on the comprehension side (19). The rapidity with which *I have a contact* emerges, however, once again results in buffered input for Parrot102 (21). Thus, linguistic processing continues in the wing agent after the lead has already begun to wait for a reply (not shown). As a final observation, note that the same u-constructor that processes *Parrot101* in Figure 6 also processes *I* in Figure 7 (u-constructor2). This is an example of where the granularity of the top-level operators affords some transfer of syntactic processing despite the difference in semantics (s-constructor6 vs. s-construct26).

5 Conclusions

The ability to communicate in natural language can be a key capability contributing to an IFOR's performance in both simulation and training exercises. In this paper we have discussed how the design of NL-Soar uses linear complexity, interruptibility, and atomaticity of language processing to provide a language capability that does not compromise reactivity. What we have not discussed, however, is the third area of interest identified in

[Rubinoff and Lehman, 1994a]: performance in accordance with empirical data from pilots in real-life simulations. Our continued work, therefore, will focus on making the NL-Soar integration more robust, including handling linguistic constructions specific to the domain and allowing for the interruptions and self-corrections that necessarily come with real language use.

6 Acknowledgement

This research was supported under subcontract to Carnegie Mellon University from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL). The authors would like to thank BMH Associates, Inc. for their technical assistance, and gratefully acknowledge the system-building support of Greg Nelson.

References

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Laird *et al.*, 1995] John E. Laird, W. Lewis Johnson, Randolph M. Jones, Frank Koss, Jill F. Lehman, Paul E. Nielsen, Paul S. Rosenbloom, Robert Rubinoff, Karl Schwamb, Milind Tambe, Julie Van Dyke, Michael van Lent, and III Robert E. Wray. Simulated intelligent forces for air: The soar/ifor project 1995. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, 1995.

[Lehman *et al.*, 1991a] J. Fain Lehman, R. Lewis, and A. Newell. Integrating knowledge sources in language comprehension. In *Proceedings of the Thirteenth Annual Conferences of the Cognitive Science Society*, 1991.

[Lehman *et al.*, 1991b] J. Fain Lehman, R. Lewis, and A. Newell. Natural language comprehension in soar: Spring 1991. Technical report, School of Computer Science, Carnegie Mellon University, CMU-CS-91-117, 1991.

[Lehman *et al.*, 1996] J. Fain Lehman, R. Lewis, and A. Newell. NL-Soar: Architectural influences on language comprehension. In *Cognitive Architecture*. Ablex Press, 1996. in press.

[Lewis, 1993] R. L. Lewis. *An Architecturally-based Theory of Human Sentence Comprehension*. PhD thesis, Carnegie Mellon University, 1993.

[Nelson *et al.*, 1994a] G. Nelson, J. F. Lehman, and B. E. John. Experiences in interruptible language processing. In *Proceedings of the 1994 AAAI Spring Symposium on Active NLP*, 1994.

[Nelson *et al.*, 1994b] G. Nelson, J. F. Lehman, and B. E. John. Integrating cognitive capabilities in a real-time task. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 1994.

[Rubinoff and Lehman, 1994a] R. Rubinoff and J. F. Lehman. Natural language processing in an ifor pilot. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 97–104, 1994.

[Rubinoff and Lehman, 1994b] R. Rubinoff and J. F. Lehman. Real-time natural language generation in nl-soar. In *Proceedings of the Seventh International Workshop on Natural Language Generation*, 1994.

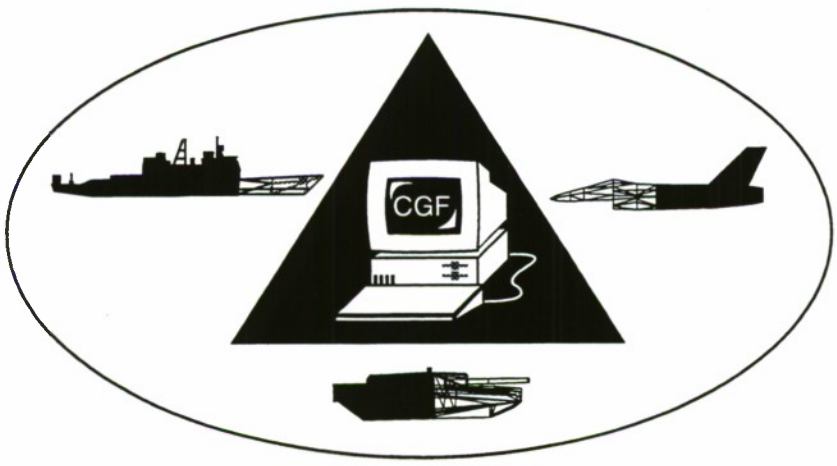
[Salisbury, 1995] M. Salisbury. Command and Control Simulation Interface Language (ccsil): Status update. In *Proceedings of the the 12th Distributed Interactive Simulation Workshop*, 1995. Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

7 Biographies

Jill Fain Lehman is a research computer scientist in Carnegie Mellon's School of Computer Science. She received her B.S. from Yale in 1981, and her M.S. and Ph.D. from Carnegie Mellon in 1987 and 1989, respectively. Her research interests span the area of natural language processing: comprehension and generation, models of linguistic performance, and machine learning techniques for language acquisition. Her main project is NL-Soar, the natural language effort within the Soar project.

Robert Rubinoff is a postdoctoral research fellow in Carnegie Mellon's School of Computer Science. He received his B.A., M.S.E., and Ph.D. from the University of Pennsylvania in 1982, 1986, and 1992, respectively; his dissertation research was on "Negotiation, Feedback, and Perspective within Natural Language Generation". His research interests include natural language processing, knowledge representation, and reasoning. He is currently working on natural language generation within the Soar project.

Julie Van Dyke is a Research Programmer working on language comprehension in NL-Soar. She is also working toward an MS in Computational Linguistics with a focus on modeling language acquisition.


CGF

Agent Tracking in Complex Multi-agent Environments: New Results

Milind Tambe and Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
email: {tambe, rosenbloom}@isi.edu

1. Abstract

Agent tracking is an important capability that an intelligent agent requires for interacting with other agents. It involves monitoring the observable actions of other agents as well as inferring their unobserved actions or high-level plans, goals and behaviors. The dynamic, real-time, multi-agent environment of air-combat simulation presents several novel challenges for agent tracking. In particular, an intelligent pilot agent needs to track the highly flexible mix of goal-driven and reactive behaviors of other pilots, track in real-time with sufficient accuracy despite ambiguities, track the activities of groups of pilots, and recursively track its own activities and possibly those of other pilots.

In our previous report at this conference (Tambe and Rosenbloom, 1994), we presented an approach that enabled a pilot agent to track flexible and reactive behaviors. This paper presents new results that address some of the remaining challenges via intra-model and inter-model optimization techniques. We have developed a system based on these techniques, and present some experimental results from it.

2. Introduction

The Soar/IFOR project (Laird et al., 1995, Rosenbloom, et al., 1994, Tambe et al., 1995) has been developing intelligent pilot agents (henceforth IPs) for participation in simulated battlefield exercises intended for training as well as for testing of new doctrine, tactics and weapon system concepts (Loper et al., 1994). These IPs have already participated in simulated combat exercises with expert human pilots, including the STOW-E exercise in November, 1994, a precursor to the much larger scale STOW-97 exercise. *Agent tracking* is one key capability that the IPs require for effective participation in these exercises. It involves monitoring the observable actions of other agents as well as inferring their unobserved actions, high-level goals, plans and behaviors. This capability is particularly useful in simulated air-combat to track the activities of other hostile or friendly pilots.

The example air-combat simulation scenario in Figure 1, based on (Tambe and Rosenbloom, 1994), illustrates the importance of agent tracking. It shows two combatting IPs: **L** in the light-shaded aircraft and **D** in the dark-shaded one. Initially, **L** and **D**'s aircraft are 50 miles apart, so they can only see each other's actions on radar. For effective performance, they have to continually track these actions. Indeed, **D** is able to survive a missile attack by **L** in this scenario due to such tracking, despite the missile being invisible to **D**'s radar.

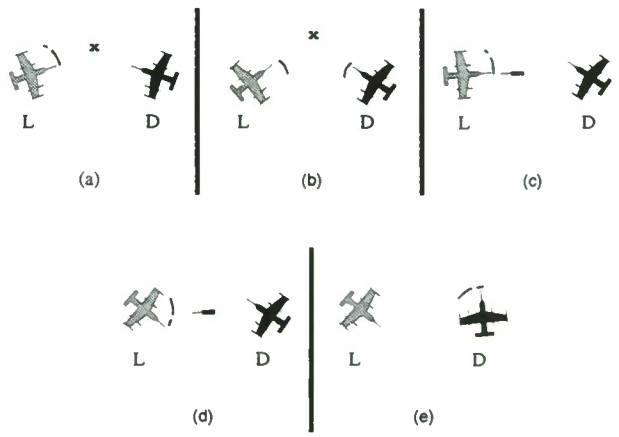


Figure 1: A simulated air-combat scenario from (Tambe and Rosenbloom, 1994). An arc on an aircraft's nose shows its turn direction.

In Figure 1-a, **D** observes **L** turning its aircraft to a collision-course heading (i.e., at this heading, **L** will collide with **D** at the point shown by **x**). Since this heading is often used to reach one's missile firing range, **D** infers the possibility that **L** is trying to reach this range to fire a missile. In Figure 1-b, **D** turns its aircraft 15° right. **L** reacts by turning 15° left, to maintain collision course. In Figure 1-c, **L** reaches its missile range, points its aircraft at **D**'s aircraft and fires a radar-guided missile. While **D** cannot see the missile on its radar, it observes **L**'s turn, and infers it to be part of **L**'s missile firing behavior. Subsequently, **D** observes **L** executing a 35° turn away from its aircraft (Figure 1-d). **D** infers this to be an *fpole* turn, typically executed after firing a missile to provide radar guidance to the missile, while slowing the closure between the two aircraft. While **D** still cannot observe the missile, it is now sufficiently convinced to attempt to evade the missile

by turning 90° relative to the direction of **L**'s aircraft (Figure 1-e). This *beam* turn causes **D**'s aircraft to become invisible to **L**'s (doppler) radar. Deprived of radar guidance, **L**'s missile is rendered harmless. Meanwhile, **L** tracks **D**'s beam turn in Figure 1-e, and prepares counter-measures in anticipation of the likely loss of both its missile and radar contact.

Thus, IPs need to continually track their opponents' actions, such as turns, and infer unobserved actions, high-level goals and behaviors, such as the fpole, beam or missile firing behaviors. This agent tracking capability is related to plan recognition (Kautz and Allen, 1986, Azarewicz, et al., 1986, Song and Cohen, 1991, Carberry, 1990), which involves recognizing agents' plans based on observations of their actions. Indeed, as with plan recognition, one key problem in agent tracking is resolving ambiguities. For instance, when **L** turns in Figure 1-c, from **D**'s perspective, it could be a turn to fire a missile, but it could also be the beginning of a 180° turn to run away, or a turn to re-establish contact given a problem in **L**'s radar. To accurately track **L**'s maneuvers **D** has to resolve such ambiguities. Despite such similarities, as we reported in the proceedings of this conference last year (Tambe and Rosenbloom, 1994) (also see (Tambe and Rosenbloom, 1995a)), agent tracking in the air-combat simulation environment brings up a novel combination of challenges. In particular, previous plan recognition work has primary focused on static, single-agent environments where agents tend to adhere to rigid plans. In stark contrast, the air-combat simulation environment is a real-time, dynamic, multi-agent environment. It is also a real-world environment, with realistic sensors,[1] and complex agent behaviors. The key issues that arise as a result are:

1. IPs must track other pilots' highly flexible mix of goal-driven and reactive behaviors, rather than purely plan-based behaviors (as with the previous work on plan recognition).
2. IPs must track opponents' maneuvers in real-time with sufficient accuracy, despite ambiguities.
3. The interactions among IPs create a need for recursive agent tracking — an IP needs to recursively track its opponent's tracking of its own maneuvers so as to understand their impact on the opponent.
4. IPs need to track maneuvers of groups of opponents.

In (Tambe and Rosenbloom, 1994), we presented an approach to agent tracking that addresses the first issue above of tracking flexible and reactive behaviors. The approach was based on the *model tracing* technique used in intelligent tutoring systems (ITS) for tracking student actions (Anderson, et al., 1990, Ward, 1991). Thus, to track the activities of its opponent, an IP executes a model of that opponent to generate predictions matching actual observations of that opponent's actions. However, as with plan recognition, previous ITS work has primarily focused on static environments.[2] This work applies model tracing to the dynamic, multi-agent environment of air-combat simulation, where agents exhibit a complex mix of goal-driven and reactive behaviors. Section 3 presents an overview of this work.

In this paper, we present new results pertaining to the three issues above that were left unaddressed in our previous report.[3] The first unaddressed issue concerns real-time tracking with accurate resolution of ambiguities. Previous model tracing and plan recognition systems have certainly dealt with the problem of ambiguity resolution; however, their solutions have turned out to be inappropriate, because of the real-time nature of this domain. In Section 4, we present a new approach called RESC (for *REal-time Situated Commitments*), that builds upon our previous work, and enables ambiguity resolution within real-time constraints. RESC's situatedness is based on its ability to continuously track an opponent's actions in the current world state. Despite the ambiguities it faces, RESC quickly commits to a single interpretation of the opponent's on-going actions — there is no exhaustive examination of alternatives. With additional information becoming available later, these commitments may turn out to be inappropriate, and the interpretations may need to be revised. These revisions occur on-line, in the context of the current state via a process called *single-state backtracking*. RESC's real-time character derives from its situatedness, its quick commitments, and its single-state backtracking.

Section 5 applies the RESC approach to address the remaining two issues from the list: recursive agent tracking and agent-group tracking. Recursive

[1]While the simulated radar, which is the primary sensor of concern here provides numbers as input instead of radar images, it does realistically model a radar's limitations.

[2]While there are some recent ITS applications that have ventured into dynamic environments, e.g., REACT (Hill and Johnson, 1994), they still primarily rely upon a plan-driven tracking strategy, dealing with the dynamic aspects as exceptions.

[3]This is an overview of results presented elsewhere in (Tambe and Rosenbloom, 1995a), (Tambe and Rosenbloom, 1995b) and (Tambe, 1995).

agent tracking requires that an IP execute its model of its opponent's model of itself. For example, to engage in recursive agent tracking, **D** would execute its model of **L**'s model of **D**. Tracking agent-groups requires that an IP execute its models of the different opponents. Unfortunately, the recursive tracking of a large opponent group can involve the execution of a large number of models. Given N opponents, and *r* levels of nested models, an IP may need to track an exponential number ($O(N^r)$) of models. Since this computational cost is unacceptable in this real-time environment, the IP uses heuristic optimizations of *model selectivity* and *model sharing* to reduce the number of models it has to execute. In contrast with RESC, which is an *intra-model* approach to real-time tracking, model selectivity and model sharing are essential *inter-model* optimizations in service of real-time tracking. These inter-model optimizations are independent of the underlying intra-model mechanisms.

All of the descriptions in this paper are provided in concrete terms, using an implementation of the IPs in TacAir-Soar^{RESC}, which is an experimental variant of the TacAir-Soar system created as part of the Soar/IFOR project (Tambe et al., 1995, Rosenbloom, et al., 1994, Laird et al., 1995). The TacAir-Soar^{RESC} implementation will also be used in Section 6 to present an empirical evaluation of the effectiveness of the RESC approach, and the inter-model optimizations. TacAir-Soar^{RESC} contains about 1050 rules, which is about half the number of rules in TacAir-Soar. The main motivation behind TacAir-Soar^{RESC} is to understand the principles underlying agent tracking. To that end, it aggressively engages in agent tracking, using experimental methods such as RESC, while TacAir-Soar takes a more conservative approach. There are thus some differences between the two systems with respect to agent tracking, although TacAir-Soar's agent tracking capabilities are a strict subset of the agent tracking capabilities of TacAir-Soar^{RESC}. Proven agent-tracking techniques uncovered in TacAir-Soar^{RESC} are to be (and some already have been) transferred back into TacAir-Soar.

As with TacAir-Soar, TacAir-Soar^{RESC} is built using the Soar architecture (Laird, Newell, and Rosenbloom, 1987, Rosenbloom, et al., 1991). In our descriptions, we will assume some familiarity with Soar's problem-solving model, which involves applying operators to states to reach a desired state.

3. Tracking Flexible Behaviors

The basic approach for tracking an opponent's actions is to execute a model of that opponent, while matching the model's predictions against the opponent's actual actions. Thus, to track its opponent **L**'s actions, **D** executes a model of **L**, matching predictions against actions. The key observation here, that enables **D** to meet the challenge of tracking flexible and reactive behaviors is the following: **D** and **L**, and in general other IPs involved in air-combat, exhibit a similar flexible mix of goal-driven and reactive behavior. Thus, while executing **L**'s model, **D** reuses the mechanisms that it uses in generating its own flexible and reactive behaviors.

To understand this in more detail, consider **D**'s internal state and operator hierarchy as depicted in Figure 2-a. **D**'s internal state maintains information regarding mission specifications, and receives input from its radar or visual sensor. Based on the state, **D** makes appropriate operator selections so as to generate the desired behavior in its external environment. In this case, it has selected *execute-mission* as the top-most operator. Since the termination condition of this operator — completion of **D**'s mission — is not yet achieved, a subgoal is generated. The *intercept* operator is selected in that subgoal. In the following subgoal, the *employ-missile* operator is selected. The subgoal after that applies *get-firing-position* to get to a missile firing position. Skipping down to the final subgoal, *maintain-heading* enables **D** to maintain heading, as seen in Figure 1-c. The operators in Figure 2-a used for generating **D**'s own actions will henceforth be denoted with the subscript **D**, e.g., $intercept_{\mathbf{D}}$. $Operator_{\mathbf{D}}$ will denote an arbitrary operator of **D**. $State_{\mathbf{D}}$ will denote the state. Together, $state_{\mathbf{D}}$ and the $operator_{\mathbf{D}}$ hierarchy may be considered as **D**'s model of its present self, referred to as $model_{\mathbf{D}}$.

$Model_{\mathbf{D}}$ supports **D**'s flexible/reactive behavior given the mechanisms for operator selection and termination in the Soar architecture. These mechanisms include Soar's preference-based decision procedure for operator selection, and termination conditions for operators (Laird and Rosenbloom, 1990). To illustrate the reuse of these mechanisms for tracking, we assume for now that **D** and **L** possess an identical set of maneuvers.[4] Thus, **D** uses a hierarchy such as the one in Figure 2-b to track **L**'s behaviors. Here, the hierarchy represents **D**'s model of **L**'s current operators in the situation of Figure 1-c. These operators are denoted with the subscript **DL**. The $operator_{\mathbf{DL}}$ hierarchy, along with the $state_{\mathbf{DL}}$ that goes with it, constitute **D**'s model of **L** or $model_{\mathbf{DL}}$. $Model_{\mathbf{DL}}$ obviously cannot and does not directly influence **L**'s actual behavior, it only tracks **L**'s behavior. For instance, in the final

[4]This is not a necessary condition. The main requirement is for an accurate model of the possible maneuvers.

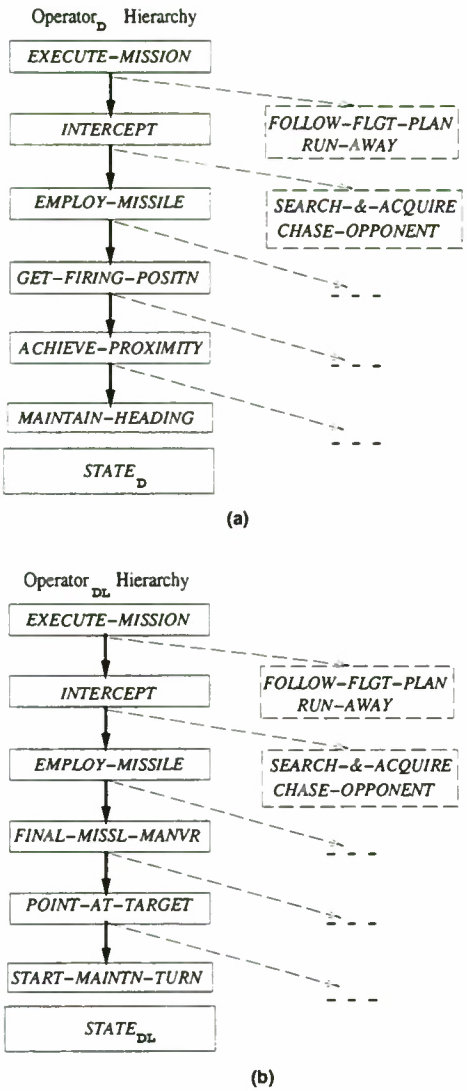


Figure 2: (a) Model$_D$, and (b) Model$_{DL}$. Solid lines indicate the actual operator hierarchy; dashed lines indicate unselected alternatives. (e.g., *run-away* is an alternative to *intercept*.)

subgoal **D** has applied the *start-&-maintain-turn*$_{DL}$ operator to state$_{DL}$. This operator cannot cause **L** to turn. It predicts **L**'s turn and matches the prediction against **L**'s actual action. Thus, if **L** starts turning to point at **D**'s aircraft, then there is a match with model$_{DL}$'s predictions. Given this match, **D** now believes **L** is turning to point at its target, (i.e., **D**), to fire a missile, as indicated by other higher-level operators in the hierarchy. **D** tracks **L**'s behaviors in this manner by continuously executing the operator$_{DL}$ hierarchy, and matching it with **L**'s actions.

Thus, **D** uses a uniform apparatus for the generation of its own flexible/reactive behaviors and tracking other agents' behaviors. In particular, operator$_D$ and operator$_{DL}$ are selected and terminated in the same flexible manner. For instance, as state$_{DL}$ changes, an operator$_{DL}$ terminates if its conditions are satisfied, and new operators$_{DL}$ get selected. This enables **D** to track **L**'s on-going flexible and reactive behaviors.

To engage in its own actions while simultaneously tracking its opponent's actions, **D** needs to execute both model$_D$ and model$_{DL}$ simultaneously. It currently achieves this by cycling through the process of selecting an operator randomly from the heap of applicable operators$_D$ and operators$_{DL}$, and executing that. As discussed later, if there are more models, the heap of available operators can grow quite large, and the random selection strategy can prove inadequate. A better operator selection strategy is an issue for future work.

4. Addressing Ambiguity in Real-time

There are two types of ambiguities in the agent tracking process introduced in the previous section. The first type is the alternative tracking operators, as shown in the dashed boxes in Figure 2-b. Given this ambiguity, it is possible for **D** to make an inaccurate selection of an operator$_{DL}$. An inaccurate selection typically results in a *match failure*, i.e., a difference in the anticipated and actual observed action. For example, the operators in Figure 2-b predict **L** will turn to point at **D**'s aircraft. If **L** were to turn away from **D**, there would be a match failure. The second type of ambiguity is seen in state$_{DL}$. **D** needs to model on state$_{DL}$ ambiguous static information, such as **L**'s radar and missile capabilities, and equally ambiguous dynamic information, such as whether **L** has detected **D** on radar. Inaccuracies in state$_{DL}$ may result in an incorrect operator$_{DL}$ being selected, and thus may also result in match failures. This paper will mostly focus on the first type of ambiguity; see (Tambe and Rosenbloom, 1995a) for more details on resolving the second type of ambiguity.

Ambiguity resolution has been the focus of much previous work in plan recognition and model tracing. The challenge here, not addressed in previous work, is that an IP has to resolve ambiguity in real-time. If an IP lags behind in tracking, it risks being ignorant of an opponent's important maneuvers that may either jeopardize its own survival, or otherwise deprive it of gaining an advantage over its opponent. Ambiguity resolution techniques that have been previously suggested in the literature, such as an exhaustive search of alternatives (Ward, 1991), or automated deduction (Kautz and Allen, 1986), fail to meet this real-time challenge. An IP also cannot avoid the ambiguity resolution problem by relying on an abstract characterization of the opponent's actions.

For instance, **D**'s recognition that **L** is engaged in an *intercept* may be accurate and unambiguous, but it is not sufficiently specific to enable **D** to counter-act **L**'s maneuvers by, say, evading a missile fired at it.

To meet the challenge of real-time ambiguity resolution, we propose a new approach called RESC (REal-time Situated Commitments). RESC relies on three sub-components to meet this real-time challenge: (i) situatedness, (ii) quick commitments, and (iii) single-state backtracking.

RESC's situatedness arises from its continuous tracking of **L**'s on-going actions and behaviors in the context of the current state_{DL}. Thus, as state_{DL} changes, it can cause an $\text{operator}_{\text{DL}}$ to terminate by satisfying its termination conditions, and cause new $\text{operator}_{\text{DL}}$ to be selected.

RESC's commitment is to a single model_{DL}, with a single state_{DL} that records the on-going world situation in real-time, and a single $\text{operator}_{\text{DL}}$ hierarchy, that provides an on-going interpretation of an opponent's actions. Given the intense real-time pressure, RESC does not spend time trying to match alternatives. Instead, it applies three inexpensive filters to weed out unsuitable alternatives. The first *fuzz box filter* filters out small changes in **L**'s heading, since these are typically small errors in an IP's actions, and tracking such detailed errors is unuseful. The second *bottom up filter* uses observations of an opponent's actions to avoid generating alternatives that are guaranteed to lead to match failures. For instance, if **L** is turning right, any alternative $\text{operator}_{\text{DL}}$ that models **L**'s left turn is guaranteed to cause match failure, and need not be generated. A third *worst case filter*, applied after the first two, selects the worst of the remaining alternatives — since this is a hostile environment, an opponent is likely to engage in the most harmful maneuver. For instance, if there is ambiguity between $\textit{run-away}_{\text{DL}}$ or $\textit{intercept}_{\text{DL}}$, **D** will select the more harmful $\textit{intercept}_{\text{DL}}$. If more than one alternative still remain active, **D** will pick one at random and commit to that.

When faced with ambiguity, it is possible that RESC commits to an inaccurate $\text{operator}_{\text{DL}}$ and state_{DL}, leading to a match failure. RESC recovers from such failures by relying on single-state backtracking to undo some of its commitments, resulting in the generation of new $\text{operator}_{\text{DL}}$ hierarchies, in real-time. As its name suggests, backtracking takes place within the context of a single current state_{DL}. Starting from the bottom of the $\text{operator}_{\text{DL}}$ hierarchy, operators are terminated one by one in an attempt to get alternatives to take their place. Only those alternatives that are relevant to the current state get installed in the hierarchy. Some of these alternatives may lead to match failures (possibly after changing state_{DL}). These are also replaced, until some alternative leads to an $\text{operator}_{\text{DL}}$ hierarchy that leads to match success. The key to the real-time character of this process is that it occurs in the *now*, i.e., within the context of the single updated current state. There is no re-examination of past states as would be done in normal backtrack search. Backtracking also does not remain wedded to the state where the failure occurred. Instead, it marches forward with the current continuously changing state.

To understand why such backtracking may actually work, let us consider the following example. Suppose at time T_0 **D** has committed to the model_{DL} in Figure 2-b. At time T_0, $\textit{point-at-target}_{\text{DL}}$ has match success in that, as predicted, **L** indeed starts turning towards **D**. This match success continues for some time after T_0, and state_{DL} is continuously updated with new aircraft positions during all this time. However, **L** really has decided to run away; so it continues turning 180° without stopping when pointing at **D**. This causes a match failure in the $\text{operator}_{\text{DL}}$ hierarchy. Single-state backtracking now ensues, terminating operators beginning from the bottom of the hierarchy. Finally, at time $T_0 + \delta$, $\textit{intercept}_{\text{DL}}$ is terminated and replaced by $\textit{run-away}_{\text{DL}}$. This predicts **L** to be turning towards its home-base, which successfully matches **L**'s actions.

In a normal backtrack search, in order to apply $\textit{run-away}_{\text{DL}}$ **D** would need to mentally recreate state_{DL} at time T_0 — after all, that is the state in which **L** initiated the run-away maneuver. However, with single state backtracking, **D** applies $\textit{run-away}_{\text{DL}}$ to state_{DL} as it exists at time $T_0 + \delta$. Thus, the conditions and actions of $\textit{run-away}_{\text{DL}}$ have to be such that they are independent of the passage of time δ. This is achieved here, since $\textit{run-away}_{\text{DL}}$ requires that an agent turn towards its home-base, which does not change position in δ. In fact, conditions and actions of $\text{operators}_{\text{DL}}$ typically refer to aspects of the world that change at a rate slower than the possible time delay caused in noticing a match failure and backtracking, and thus can be successfully applied in single-state backtracking.

Overall, RESC trades off completeness in service of real-time tracking. That is, RESC could potentially lead to errors in tracking, which a more exhaustive tracker could possibly avoid. Fortunately, RESC's worst-case filter attempts to ensure that its tracking errors will at least not be fatal. In any event, as illustrated in Section 6, RESC has so far not suffered

along the completeness dimension.

5. Recursive Agent and Agent-Group Tracking

The RESC approach can be applied in a straightforward manner for both recursive agent tracking and agent-group tracking. Let us first focus on recursive tracking. If **D** believes **L** to be aware of **D**'s presence, **D** can recursively track its own actions from **L**'s perspective, by executing $\text{model}_{\mathbf{DLD}}$ or **D**'s model of **L**'s model of **D**. $\text{Model}_{\mathbf{DLD}}$ consists of a $\text{state}_{\mathbf{DLD}}$ and an $\text{operator}_{\mathbf{DLD}}$ hierarchy. **D** tracks $\text{model}_{\mathbf{DLD}}$ by matching $\text{operator}_{\mathbf{DLD}}$ predictions with its own actions. Thus, if **D** were to engage in an $\text{fpole}_{\mathbf{D}}$ after a missile firing, it would be **D**'s recursive tracking of $\textit{fpole}_{\mathbf{DLD}}$ which would indicate a missile firing to $\text{model}_{\mathbf{DL}}$, eventually leading to the selection of $\textit{evade-missile}_{\mathbf{DL}}$ to track **L**'s missile evasion. On the contrary, if **D** believes **L** to be unaware of itself, then it would not create a $\text{model}_{\mathbf{DLD}}$. In this case, $\text{model}_{\mathbf{DL}}$ will not be informed of **D**'s missile firing, and thus, **D** would not expect **L** to engage in $\textit{evade-missile}_{\mathbf{DL}}$. Further nesting of recursive models could lead to the creation of $\text{model}_{\mathbf{DLDL}}$, $\text{model}_{\mathbf{DLDLD}}$, etc.

To track groups of opponents, **D** can again execute its models of different individual opponents. For instance, suppose a second opponent, **J** joins **L** in attacking **D**. **D** can track **J**'s actions just as it tracked **L**'s actions, by executing new models, such as $\text{model}_{\mathbf{DJ}}$ and $\text{model}_{\mathbf{DJD}}$. In addition, it may also execute others such as $\text{model}_{\mathbf{DJL}}$, $\text{model}_{\mathbf{DLJ}}$, $\text{model}_{\mathbf{DLJL}}$ and so on. These models may be important to track **J**'s interactions with **L**.

Unfortunately, this scheme points to an exponential growth in the number of models that an IP needs to execute. In general, for N opponents, and r levels of nesting (measured with r=1 for $\text{model}_{\mathbf{D}}$, r=2 for $\text{model}_{\mathbf{DL}}$, and so on), **L** may need to execute $\sum_{i=0}^{r-1} N^i = \frac{N^r-1}{N-1}$ different models. Given its limited computational resources, **D** would be unable to execute relevant operators from all these models in real-time, jeopardizing its combat effectiveness and survival.

Two inter-model optimizations, *model selectivity* and *model sharing*, can help alleviate this problem. Model selectivity suggests selective construction of only those models as are relevant to the given situation. Such selectivity may be exercised based on the current or anticipated level of interaction between two agents, and abstraction. For instance, suppose **D** determines the two opponents attacking it, **L** and **J** to be independent of each other. It can then avoid the construction of models such as $\text{model}_{\mathbf{DLJ}}$ and $\text{model}_{\mathbf{DJL}}$ since those were originally intended to model their interactions. For further selectivity, **D** may also abstract away from differences in recursive models across levels, at least at deep levels of nesting. Currently, based on our interviews with domain experts (pilots), we have set the level of nesting to $r \leq 3$. Thus, **D** avoids creating models at $r \geq 4$ such as $\text{model}_{\mathbf{DLDL}}$ — it abstracts away from differences in models such as $\text{model}_{\mathbf{DL}}$ and $\text{model}_{\mathbf{DLDL}}$.

The overall result of model selectivity is to reduce the total number of models for N independent opponents from $\frac{N^r-1}{N-1}$ to 2N+1. Model sharing now helps in further reduction by suggesting direct sharing of states and operators among different models, to reduce the number of models being executed. For an illustration of this optimization, consider $\text{model}_{\mathbf{D}}$ and $\text{model}_{\mathbf{DLD}}$. The $\text{operator}_{\mathbf{D}}$ hierarchy can be shared with the $\text{operator}_{\mathbf{DLD}}$ hierarchy since the two are often identical. Furthermore, information in $\text{state}_{\mathbf{D}}$, such as radar input, is shared with $\text{state}_{\mathbf{DLD}}$. If there may be some unsharable secret aspects of $\text{state}_{\mathbf{D}}$, e.g., if **D**'s missile range is a secret, then it will be maintained on $\text{state}_{\mathbf{D}}$, but it will not be shared with $\text{state}_{\mathbf{DLD}}$. With the sharing of such recursive models, the total number of models being executed goes down to N+1. There is a further opportunity for model sharing, if the opponents are seen to be attacking in a closely coordinated fashion, as often occurs in air-to-air combat. For instance, if **L** and **J** are closely coordinated, **D** may share $\text{model}_{\mathbf{DL}}$ with $\text{model}_{\mathbf{DJ}}$, and $\text{model}_{\mathbf{DLD}}$ with $\text{model}_{\mathbf{DJD}}$. Thus, if all N opponents are coordinated, **D** may need to execute only two models instead of N+1.

Unfortunately, in some cases, shared models need to be un-shared (that is one of the key reason why the models are shared rather than being eliminated via selectivity). For instance, **L** and **J** may initially engage in identical maneuvers, but then launch a *pincer* tactical attack on **D** from two sides. Tracking this requires that there be some unsharing of models, since **L** may be turning left as part of the pincer while **J** turns right. The penalty for such unsharing is in terms of copying information from what was previously one model into a second one. Thus, an IP has to trade off the benefits of sharing with the cost of possible unsharing. In general, if it greedily shares two models whenever they appear identical, it could face a very heavy unsharing overhead. At present, this sharing question is resolved based on the following heuristics:

- If two agents are engaged in a tightly coordinated attack, then their models are shared. If they engage in a tactic, such as a pincer, that requires model unsharing, then the models are not re-shared. The motivation being that if the models are unshared once, they are likely to face unsharing repeatedly, and thus, re-sharing would not be beneficial.
- An agent's model and its recursive agent model (e.g., $model_{D}$ and $model_{DLD}$) are shared. When an agent engages in a deceptive tactic, the models are partially unshared. They are then re-shared upon completion of the deceptive tactic.

6. Evaluating TacAir-Soar^{RESC}

As mentioned in Section 2, we have developed a system called TacAir-Soar^{RESC} that brings together the ideas presented in this paper. There are two aspects to the evaluation of TacAir-Soar^{RESC}. The first aspect is whether the current approach enables **D**, the TacAir-Soar^{RESC} pilot agent, to track its opponents' actions accurately in real-time. We conducted two sets of experiments to address this issue. The first set involved running Soar-vs-Soar air-combat simulation scenarios, as outlined by the human experts, with the number of opponents varying from one to four.[5] The following presents some observations about these experiments, presented in terms of the TacAir-Soar^{RESC} based IP **D**'s ability to track its opponent's maneuvers:

1. The amount of effort involved in agent tracking, as measured by the percentage of overall operators involved in agent tracking varied from 8% to 63%. The actual effort depended on both the number of opponents and the complexity of the opponents' maneuvers.
2. In some scenarios up to 20% of the agent tracking operators resulted in match failures. However, the IP still recovered from these failures, and settled on a matching operator in real-time.
3. The model-sharing and selectivity optimizations do help in alleviating resource contentions. In the case of four coordinated opponents, the model-sharing optimization reduces the number of models from (2N+1 =) 9 to just 2. In terms of actual operators, it is projected to provide up to a 4 fold improvement in operator executions.
4. In the case of four independent opponents, however, **D** still does face some resource contention, and it is seen to be unable to track the actions of all of the agents in time. Further improvements in model-sharing and model-selectivity optimizations may be required to address this issue.

Our second set of experiments involved Soar-vs-ModSAF simulated air-combat scenarios. ModSAF-based (Calder et al., 1993) pilot agents are controlled by finite state machines combined with arbitrary pieces of code, and do not exhibit high behavioral flexibility. While **D** was in general successful in agent tracking in these experiments — it did recognize the maneuvers in real-time and respond to them — one interesting issue did come up. In particular, in one of the scenarios here, there was a substantial mismatch in **D**'s worst assumptions regarding its opponent's missile capabilities and the actual capabilities — **D** assumed the range to be 30 miles, even though the actual range was about 15 miles. This led to tracking failures. Dealing with model mismatch is also an issue for future work.

The second aspect to understanding the effectiveness of TacAir-Soar^{RESC} is an estimate of the impact of agent tracking on improving **D**'s overall performance. In general, this is a difficult issue to address. A quantitative estimate is difficult, since it is difficult to quantify an improvement or degradation in **D**'s overall performance. Nonetheless, we can at least list some of the types of benefits that **D** accrues from this capability. First, agent tracking is crucial for **D**'s survival. Indeed, it is based on agent tracking that **D** can recognize an opponent's missile firing behavior and evade it. Second, agent tracking improves **D**'s overall understanding of a situation, so it can act/react more intelligently. For instance, if an opponent is understood to be running away, **D** can chase it down, which would be inappropriate if the opponent is not really running away. Similarly, if **D** is about to fire a missile, and it recognizes that the opponent is also about to do the same, then it can be more tolerant of small errors in its own missile firing position so that it can fire first. Finally, agent tracking helps **D** in providing a better explanation of its behaviors to human experts. (Such an explanation capability is currently being developed (Johnson, 1994)). If human experts see **D** as performing its task with an inaccurate understanding of opponents' actions, they may not have sufficient confidence to actually use it in training.

7. Conclusion

The real-time, dynamic, multi-agent environment of air-combat simulation, poses novel challenges for agent tracking. This paper presented some new

[5] In these experiments, the opponents did not perform coordinated tactics such as pincer or double-pincer — that is left for future work.

results in agent tracking that built upon our previous work reported at this conference (Tambe and Rosenbloom, 1994). Specific contributions include: (i) an approach called RESC (Real-time Situated Commitments) that enabled tracking of flexible and reactive behaviors with real-time ambiguity resolution: (ii) extension of RESC to recursive agent and agent-group tracking; and (iii) model selectivity and model sharing optimizations to alleviate some of the overheads of a large number of models. The paper presented results from a system called TacAir-Soar^{RESC} that was built based on the ideas presented in the paper.

An important issue for future work is tracking a coordinated tactic such as a pincer. This requires that we address, among other things, the issues of model sharing and unsharing. As noted in Section 6, addressing model mismatch is another important issue for future work. We hope that resolution of these issues will lead towards an improved understanding of agent tracking, and its possible application in other domains, such as education and entertainment.

8. Acknowledgements

This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL); and as part of contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD).

9. References

Anderson, J. R., Boyle, C. F., Corbett, A. T., and Lewis, M. W. (1990) "Cognitive modeling and intelligent tutoring". *Artificial Intelligence 42* , 7-49.

Azarewicz, J., Fala, G., Fink, R., and Heithecker, C. (1986) Plan recognition for airborne tactical decision making. Proceedings of the National Conference on Artificial Intelligence. , pp. 805-811.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z. (1993) ModSAF behavior simulation and control. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Carberry, S. (1990) Incorporating default inferences into Plan Recognition. Proceedings of National Conference on Artificial Intelligence. , pp. 471-478.

Hill, R., and Johnson, W. L. (1994) Situated plan attribution for intelligent tutoring. Proceedings of the National Conference on Artificial Intelligence.

Johnson, W. L. (August, 1994) Agents that learn to explain themselves. Proceedings of the National Conference on Artificial Intelligence. Seattle, WA,

Kautz, A., and Allen J. F. (1986) Generalized plan recognition. Proceedings of the National Conference on Artificial Intelligence. , pp. 32-37.

Laird, J.E. and Rosenbloom, P.S. (July, 1990) Integrating execution, planning, and learning in Soar for external environments. Proceedings of the National Conference on Artificial Intelligence.

Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K., Tambe, M., van Lent, M., and Wray, R., (May, 1995) Simulated Intelligent Forces for Air: The Soar/IFOR project 1995. Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.

Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987) "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 , 1-64.

The DIS steering committee. (May, 1994)The DIS vision: A map to the future of distributed simulation. Tech. Rept. IST-SP-94-01, Institute for simulation and training, University of Central Florida, .

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. (1991) "A preliminary analysis of the Soar architecture as a basis for general intelligence". *Artificial Intelligence 47*, 1-3 , 289-325.

Rosenbloom, P., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Lehman, J. F., Rubinoff, R., Schwamb, K., and Tambe, M. (1994) Intelligent Automated Agents for Tactical Air Simulation: A Progress Report. Proceedings of the Conference on Computer Generated Forces and Behavioral Representation.

Song, F. and Cohen, R. (1991) Temporal reasoning during plan recognition. Proceedings of the National Conference on Artificial Intelligence.

Tambe, M. (1995) "Recursive agent and agent-group tracking in a real-time dynamic environment". *Information Sciences Institute, University of Southern California Unpublished note* .

Tambe M., and Rosenbloom, P. S. (May, 1994) Event tracking in complex multi-agent environments. Proceedings of the Conference on

Computer Generated Forces and Behavioral Representation.

Tambe, M. and Rosenbloom, P. S. (1995) "Event Tracking in a dynamic multi-agent environment". *Computational Intelligence (To appear)* .

Tambe, M. and Rosenbloom, P. S. (1995) "RESC: an approach to real-time, dynamic agent tracking". *Information Sciences Institute, University of Southern California Unpublished note* .

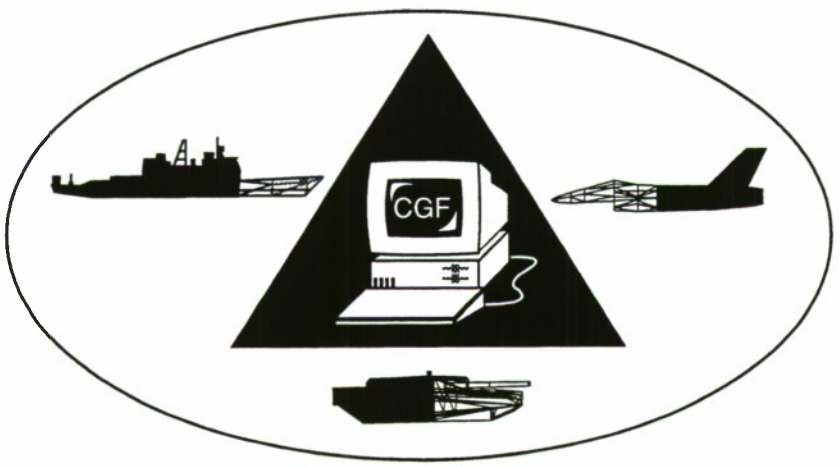
Tambe, M., Johnson, W. L., Jones, R., Koss, F., Laird, J. E., Rosenbloom, P. S., and Schwamb, K. (Spring 1995) "Intelligent agents for interactive simulation environments". *AI Magazine 16* .

Ward, B. (May 1991) *ET-Soar: Toward an ITS for Theory-Based Representations*. Ph.D. Th., School of Computer Science, Carnegie Mellon University, .

10. Authors' Biographies

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC) and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, India in 1986. He received his Ph.D. in computer science from Carnegie Mellon University in 1991. His interests are in the areas of integrated AI systems, agent modeling, plan recognition, and efficiency and scalability of AI programs, especially rule-based systems.

Paul S. Rosenbloom is an associate professor of computer science at the University of Southern California and the acting deputy director of the Intelligent Systems Division at the Information Sciences Institute. He received his B.S. degree in mathematical sciences from Stanford University in 1976 and his M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University in 1978 and 1983, respectively. His research centers on integrated intelligent systems (in particular, Soar), but also covers other areas such as machine learning, production systems, planning, and cognitive modeling. He is a Councillor and Fellow of the AAAI and a past Chair of ACM SIGART.


CGF

A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces

Michael R. Hieb, Gheorghe Tecuci, J. Mark Pullen
Department of Computer Science
George Mason University,
Fairfax, VA 22030
{hieb, tecuci, mpullen}@cs.gmu.edu

Andrew Ceranowicz
Loral Advanced Distributed
Simulation, 50 Moulton St.
Cambridge, MA 02138
aceran@camb-lads.loral.com

David Hille
ANSER
1215 Jefferson Davis Hwy
Arlington, VA 22202
hilled@anser.org

1. Abstract

The ability to build intelligent command agents for Computer Generated Forces (CGF) is significantly constrained by the knowledge acquisition effort required. Many iterations by Subject Matter Experts (SME), programmers and knowledge engineers are required to develop acceptable behavior even for a narrow range of situations. Moreover, once built the agents cannot adapt themselves to changes. This paper presents an automated knowledge acquisition system, called Captain, which allows the SME to "teach" a CGF command agent in much the same way the SME would teach a human student. Captain is built upon Disciple, a multistrategy apprenticeship learning system that combines machine learning and knowledge acquisition methods. With Captain, an SME gives the CGF command agent specific examples of problems and solutions, explanations of these solutions, or supervises the agent as it solves new problems. During such interactions, the agent learns how to behave in similar situations. This approach produces verified behavior and addresses the problem of validating existing behaviors when new behaviors are added. In this paper we describe the teaching modes of Captain and illustrate it with an extended example from a specific CGF system, Modular Semi-Automated Forces (ModSAF), where an SME teaches a CGF entity a desired behavior, using the ModSAF and Captain interfaces.

2. Introduction

We are currently developing a methodology and tool, called Captain (Tecuci, et al. 1994; Hille, et. al 1994) to construct virtual command agents for Computer Generated Forces. This general approach has the potential to significantly advance the state of the art of constructing intelligent agents. Captain is built upon the Disciple multistrategy machine learning technology we have been developing for several years (Tecuci 1988, Tecuci & Kodratoff 1990, Tecuci 1992). This methodology is based on a synergistic combination of a wide range of machine learning strategies (explanation-based learning, learning by analogy, empirical inductive learning from examples, abductive learning, etc.) which significantly increases the capabilities of a machine learning system.

An extensive and error-prone knowledge acquisition effort is currently required to develop validated acceptable behavior of CGF entities. Moreover, because CGF agents cannot adapt themselves to changes, these efforts must resume whenever tactics and weapon systems are changed, or the simulated environments become more sophisticated. New methods are needed to cope with the challenge of building virtual commanders, which can command large numbers of entity level CGF. Captain is an initial approach to this problem that enables an SME (e.g., an Armored Company Commander) to build an adaptive command agent for a CGF system, (e.g., ModSAF (Ceranowicz, 1994)), following a tutoring approach rather than a traditional knowledge engineering approach.

With Captain, an SME can teach an agent in many of the same ways the SME would teach a human student. The SME will give the agent specific examples of problems and solutions, will give explanations of these solutions, and will supervise the agent as it solves new problems. During such interactions, the agent learns general rules and concepts, continuously extending and improving its knowledge base. This approach produces verified knowledge-based command agents, because it is based on an SME interacting with, verifying and correcting the way the agent solves problems. Moreover, the command agent could continue to learn from its own experiences, during its normal use in simulations, by using the same learning strategies it employed to learn from the SME. We expect that this type of technology will be a key driver in shifting the job of agent construction from programmers and knowledge engineers to SMEs.

This paper gives an overview of the Captain methodology and tool, and illustrates them with an example of teaching an automated company commander how to place its units for a defensive mission, as in the situation in Figure 1.

The rest of the paper is organized as follows. Section 3 presents the general methodology of the Captain approach. Section 4 describes the knowledge representation of Captain. Section 5 presents the learning methods. Section 6 presents an extended example of agent training. Finally, section 7

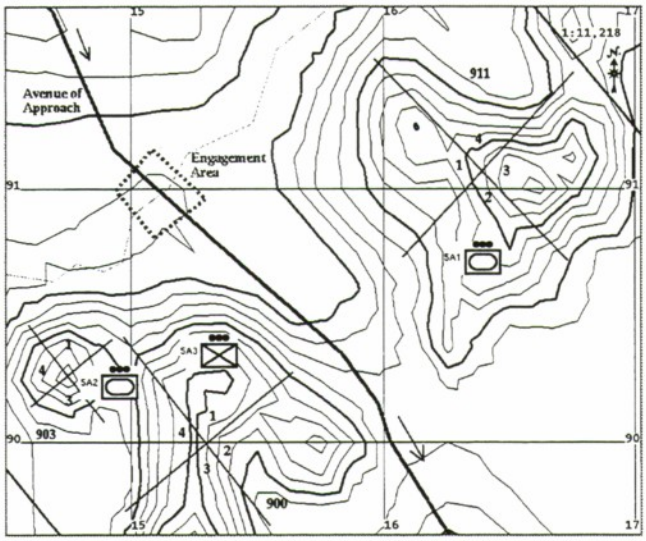


Figure 1: Placement of platoons by a Captain agent for a defensive mission

concludes the paper with a discussion of our agent-building approach.

3. Captain Methodology

CGF, such as ModSAF, provide a system for collective training of military forces that uses no fuel, causes no training injuries (except perhaps to the ego), and avoids much of the cost associated with travel to far-away training grounds. While virtual simulation will not completely replace field training, it will significantly reduce the total cost of training.

One of the applications of the Captain methodology is to build command agents at different echelons. Such command agents will generate orders for their subordinates and will represent the behavior of forces containing hundreds of vehicles, expanding the effective scope of Distributed Interactive Simulation applications to much larger forces.

In the Captain methodology, we define three phases in the creation of an agent.

In the first phase, Knowledge Elicitation, the SME works with a knowledge engineer to define an initial knowledge base (KB) which will contain whatever knowledge could be easily expressed by the expert. This KB is expected to be incomplete and partially incorrect at this point.

In the second phase, Apprenticeship Learning, the Command Agent will interactively learn from the expert by employing apprenticeship multistrategy learning. During this phase, the agent's KB is extended and corrected until it becomes complete and correct enough to meet required specifications. There are three *Interactive Learning Modes* involving an SME that are available during this phase as shown in Figure 2: Teaching, Cooperating and Critiquing.

In the *Teaching Mode*, the SME will show the agent examples of typical situations and correct orders to give to subordinate units to achieve the goals of a certain mission. This will take the form of detailing a specific scenario in ModSAF and then giving the agent a mission and specifying how it is to respond. From each such scenario, the agent will learn a rule that will allow it to respond adequately to situations similar to the one indicated by the SME. The Captain Agent will attempt to understand the example given, by asking the SME questions and asking for explanations when necessary. The Captain Agent may also elicit new concepts or relations from the SME, when it does not have the required knowledge available. An extended example of the Teaching Mode is given in Section 6.

In the *Cooperating Mode*, the Captain agent will work through a ModSAF scenario with the help of the SME. The Captain Agent will issue orders for a given mission and the SME will either verify that they are correct, or propose specific changes that the

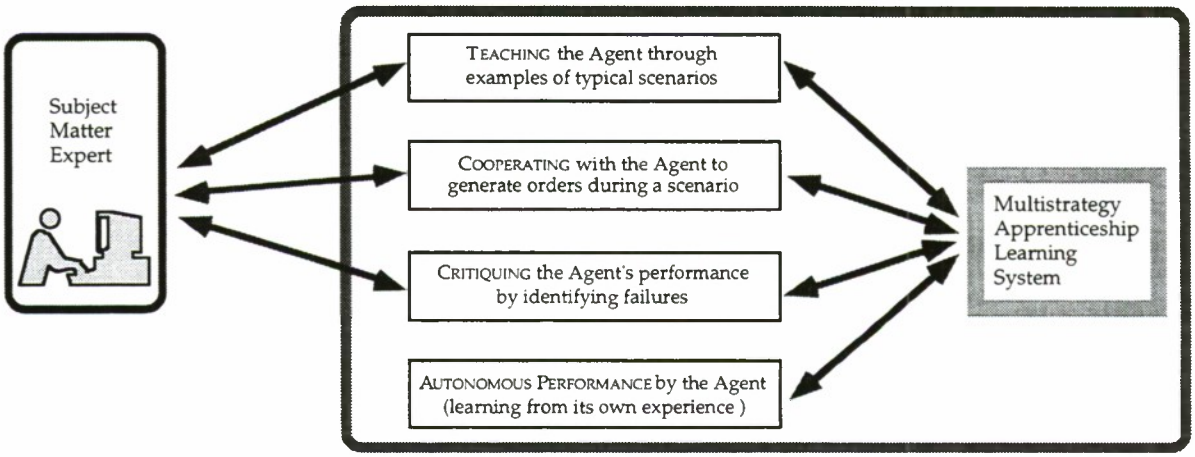


Figure 2: Different modes of learning in Captain

Captain Agent can learn from. The aim of the Cooperating Mode is to improve the rules that have already been learned. This interaction is easier for the SME than the previous mode, because the agent is generating the orders. The system will require less explanation and will not need to elicit as much new information.

In the *Critiquing Mode*, the agent will perform unassisted in a ModSAF scenario. A logger will then be used by the SME to play back the ModSAF scenario, who will select particular orders that were not generated properly, and suggest better orders. This mode could be thought of as "debugging" the learned knowledge. The aim is to verify that the learned knowledge is correct, and to improve it when necessary.

When the Captain command agent has been trained with examples of the typical ModSAF scenarios it should be able to solve, it enters a third phase, Autonomous Learning, where it is used in simulations without the assistance of the SME. However, the agent will continue to learn from its own experiences by employing the same multistrategy techniques it used when learning from an SME.

It is important to stress that both the apprenticeship learning and autonomous learning take place off-line. Therefore they are not constrained by the limited time or limited computational resources available during real-time simulations. The only constraint imposed during the actual simulation is to keep a record of the agent's decision process.

The Captain agent in ModSAF acts as an automated commander that receives missions and generates appropriate orders for its subordinate units. For instance, the agent may represent a company commander, and receive the mission of defending its area of responsibility (shown in Figure 1) against an enemy attack. The agent will then issue orders to its subordinate platoons to move to the positions indicated in Figure 1, to protect themselves and to prevent the enemy from passing through their area. Solving this placement problem requires a very complex reasoning process about terrain, weapon systems capabilities, and tactics. In the next sections we will describe the knowledge representation and reasoning of Captain agents, and how they are taught to solve the class of problems illustrated in Figure 1.

4. Knowledge Representation

Captain agents use a hybrid knowledge representation integrating semantic networks and rules. Semantic networks represent information from a terrain database at a conceptual level, as well as generic and specific knowledge about weapon systems and forces. Rules are used to represent the behavior and decision making of the agent as it generates orders for accomplishing missions.

In order to facilitate learning, the objects and the rules both use the following representation unit:

```
(concept-i concept-k    (FEATURE-1 value-1)
                                 . . .
                        (FEATURE-n value-n))
```

This expression defines 'concept-k' as being a subclass of 'concept-i' (from which it inherits features) with additional features. The value of a feature may be a constant or another concept.

Captain agents in ModSAF must reason about the placement and movement of their forces. Humans subconsciously transform terrain they see on a map (see Figure 1) or from their personal observation into an abstract model that they then use when reasoning. Automated commanders need to do a similar kind of transformation, since the data readily available to them in a terrain database has too much detail to reason about terrain efficiently. Captain uses *semantic terrain transformations* (transforming the terrain data to relevant symbolic concepts), described in (Hille, et al. 1995). After the terrain transformations are performed, the map is represented in a symbolic form expressing concepts and relationships in a semantic network. For instance, a portion of information from the map in Figure 1 is represented in the Captain knowledge base as shown in Figure 3.

The rules in the agent's knowledge base are if-then rules represented as plausible version spaces (Tecuci, 1992). A concrete example of such a rule is shown in Figure 4. It is a rule for placing the platoons of a company to defend a company's area of

```
(hill        hill-911
             (orientation "right")
             (size 5)
             (in company-a-area-of-responsibility))

(hill-sector  hill-sector-911-1
             (quadrant 1)
             (visible mobility-corridor-a)
             (visible engagement-area-a)
             (has-exit "yes")
             (in company-a-area-of-responsibility)
             (distance-to-engagement-area "close")
             (part-of hill-911))

(hill-sector  hill-sector-911-2
             (quadrant 2)
             (visible mobility-corridor-a)
             (visible engagement-area-a)
             (has-exit "yes")
             (in company-a-area-of-responsibility)
             (distance-to-engagement-area "close")
             (part-of hill-911))
```

Figure 3: Terrain knowledge

```
IF
   plausible upper bound
   (SOMETHING                AR20)
   (SOMETHING                AV21     (PART-OF AR20))
   (ENGAGEMENT-AREA          E22      (PART-OF AV21))
   (HILL-SECTOR              HS-TP1   (IN AR20) (VISIBLE E22))
   (HILL-SECTOR              HS-TP2   (IN AR20) (VISIBLE E22))
   (HILL-SECTOR              HS-I     (IN AR20) (VISIBLE E22) (DISTANCE-TO-ENGAGEMENT-AREA "close"))
   (SOMETHING                P1)
   (SOMETHING                P2)
   (SOMETHING                P3       (WEAPONS-CLASSIFICATION "light"))
   (SOMETHING                C19      (NUMBER-OF-PLATOONS 3) (COMMANDS P3) (COMMANDS P2) (COMMANDS P1))
   (DEFEND-AREA-MISSION      M23      (WITH C19) (IN AR20))

   plausible lower bound
   (AREA-OF-RESPONSIBILITY   AR20)
   (AVENUE-OF-APPROACH       AV21     (PART-OF AR20))
   (ENGAGEMENT-AREA          E22      (PART-OF AV21))
   (HILL-SECTOR              HS-TP1   (IN AR20) (VISIBLE E22))
   (HILL-SECTOR              HS-TP2   (IN AR20) (VISIBLE E22))
   (HILL-SECTOR              HS-I     (IN AR20) (VISIBLE E22) (DISTANCE-TO-ENGAGEMENT-AREA "close"))
   (ARMORED-PLATOON          P1)
   (ARMORED-PLATOON          P2)
   (INFANTRY-PLATOON         P3       (WEAPONS-CLASSIFICATION "light"))
   (COMPANY                  C19      (NUMBER-OF-PLATOONS 3) (COMMANDS P3) (COMMANDS P2) (COMMANDS P1))
   (DEFEND-AREA-MISSION      M23      (WITH C19) (IN AR20))

THEN
the problem
  PLACE-COMPANY   C19   IN AR20   TO-DESTROY-ENEMY-IN E22   FOR M23

has the following solution
  PLACE-INFANTRY-PLATOON   P3   IN   HS-I
  PLACE-TANK-PLATOON       P1   IN   HS-TP1
  PLACE-TANK-PLATOON       P2   IN   HS-TP2
```

Figure 4: A rule for company defensive placement

responsibility while protecting itself. The plausible lower bound is a conjunctive expression that is approximately less general than the hypothetical exact condition of the rule. The plausible upper bound is a conjunctive expression that is approximately more general than the hypothetical exact condition. This type of rule allows the agent to respond to a wide variety of situations.

If the plausible lower bound of the rule in Figure 4 matches the current situation, then the placement indicated by the rule is most likely a correct one. If the plausible lower bound does not match the current situation, but the plausible upper bound does, then the placement indicated by the rule is considered only plausibly correct. Finally, if the plausible upper bound does not match the current situation, the rule is not considered applicable. The placement indicated in Figure 1 was generated by applying the plausible lower bound of the rule in Figure 4, which gives this solution a high degree of confidence.

While Captain uses a plausible version space to learn rules that have plausible bounds, these plausible version space rules may then be translated to single condition rules, or other formats required by specific CGF systems.

It is important to stress that there are many correct placements, corresponding to different ways of matching the plausible lower bound and the situation represented in Figure 1. For instance, the 1st armored platoon could also be placed in sector 1 of hill 911. During a simulation, the agent may randomly pick one placement covered by the lower bound. This is a very important aspect of our rule representation that accounts for the unpredictability of an agent's behavior, a feature which is characteristic of human agents but very rare among automated agents. The rules from the agent's knowledge base are learned by the agent during training sessions with a human expert, as illustrated in Section 6. The generalization language for the rules is provided by the semantic network of object concepts. This language is incomplete and partially incorrect, and will also evolve during the training sessions. Therefore, the learning goal of the agent is not to learn a consistent and complete rule, but a rule that has as few inconsistencies as possible and is as complete as possible.

5. Learning in Captain

The rule learning process is summarized in Table 1 and partially illustrated in the next section which presents a training session with Captain and ModSAF in which Captain interacts directly with an SME to learn how to place platoons of a ModSAF company commander to defend the company's area of responsibility. An example of such a scenario is shown in Figure 1.

Given a company and its area of responsibility, the SME is asked to place the company's platoons using the ModSAF graphical interface. The SME is then asked to select relevant explanations from a menu of plausible explanations generated by Captain. Captain generates these explanations by using heuristics to look for various types of correlations in its knowledge of the situation.

Given knowledge of the current situation, the correct placement for the defensive mission and the explanations of why the placement is good, Captain generates an initial plausible version space rule for placing platoons. However the rule is not completely learned and is still in a very general form that allows a great many possible placements, some of which may not be correct. The system uses this intermediate form of the rule to generate a concrete placement for the company and shows this to the SME on the ModSAF graphical display. The SME is asked if this is a good placement. If the SME agrees, then the system generalizes from the two placements utilized and learns a better rule. If the SME disagrees then the system elicits an explanation as to why the placement is not correct (e.g., one of the platoons cannot see the area of engagement or the infantry is too far away) and corrects the rule to eliminate the generation of such placements. The learning process continues in this manner. After Captain has learned a rule, the SME is allowed to validate the rule by examining other placements which Captain generates. When the SME is satisfied that the rule represents acceptable behavior the rule is placed into the Captain knowledge base. As noted previously, the rule(s) learned may then be translated into forms required by specific CGF systems.

During learning, both the plausible lower bound and the plausible upper bound are adjusted to better approximate the hypothetical exact condition of the rule. This is achieved by successive generalizations and specializations of each of the bounds. However, in spite of these incremental adjustments, the plausible bounds may not become identical because of the incompleteness of the representation language or because there are not enough examples to learn from.

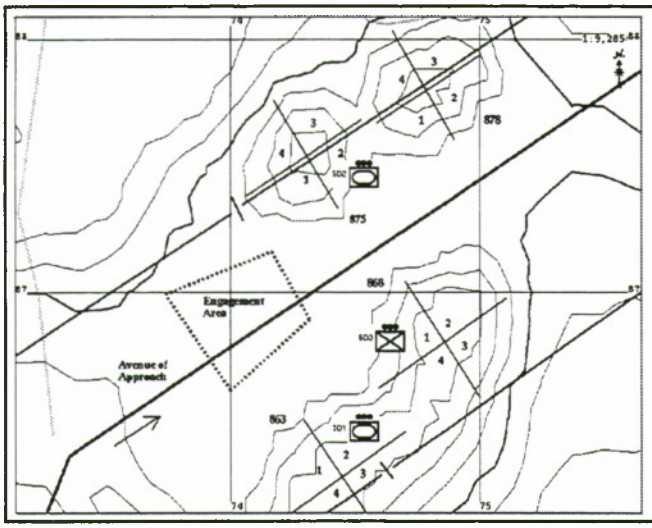
The SME may express whatever knowledge is necessary at any time in the process. If an explanation is not given when the initial rule is formed, it may be given at a later time. In this way, the rule for unit placement is incrementally improved during the teaching mode. The rule is available for modification at any time, even by another SME. The teaching mode stops when the SME is satisfied with the behavior, or the system has considered all possible solutions to the problem presented. After the initial rule is learned in the Teaching Mode, it may be further refined and verified in the subsequent two modes, Cooperating and Critiquing. In these modes the placement rule would be used by the agent in the performance of an actual mission.

INPUT: an example of problem and its correct solution indicated by the expert
• Find an explanation of the validity of the training example
• Define an initial plausible version space for the rule to be learned, in which the plausible lower bound corresponds to the explanation of the training example, and the plausible upper bound corresponds to an over-generalization of the explanation
• Use the plausible upper bound to generate examples analogous with the input and ask the expert to characterize each of them as correct (positive example) or incorrect (negative example). Additional examples may also be indicated by the expert.
• Use these analogous examples to elicit additional explanations from the expert and to modify the plausible bounds of the rule's condition to better approximate the hypothetical exact condition.
OUTPUT: one or more rules for solving the problems illustrated by the initial example, as well as an improved semantic network representing more complete and correct descriptions of the application domain objects.

Table 1: Overview of the learning method

6 An Illustration of Teaching a Captain Agent

In this section we will illustrate how an SME will teach a Captain agent to behave as an automated company commander. We will show how the SME will teach the Captain agent how to place its units in order to defend an area from an enemy attack. The SME initiates the teaching session by showing the agent a specific example of a correct placement. The SME places the three platoons of Company D on the ModSAF map, to defend the company's area of responsibility, as indicated in the left hand side of Figure 5. The SME uses the ModSAF simulation interface as the SME normally would when orienting units. The right hand side of Figure 5 shows the textual representation of the example mission and



Initial Problem:

Place-company company-D
in company-D-area-of-responsibility
to-destroy-enemy-in engagement-area-D
for defend-area-mission

Initial Solution:

place-infantry-platoon platoon-D3
in hill-sector-868-1
position-tank-platoon platoon-D1
in hill-sector-863-2
position-tank-platoon platoon-D2
in hill-sector-875-2

Figure 5: Initial Placement for Company D

also the solution. The system maintains a correspondence between each concept in the textual representation (e.g., hill-sector-868-1) and the corresponding object (region) on the map.

The Captain agent will attempt to understand why the indicated solution is correct. It will use several heuristics to propose partial plausible explanations from which the SME has to choose the relevant ones, as indicated in Figure 6. The partial explanations proposed by the system are relationships between the objects from the problem and its solution, or properties of these objects that are represented in the agent's knowledge base. For instance, in the case of the example considered, they are relationships between the platoons and the terrain features.

There are several general explanation patterns in Captain, which are matched against the knowledge base to generate specific plausible explanations. The SME first selects the relevant ones, then may request additional explanations, or the SME may give additional relevant explanations (which were not generated by the system). As indicated in Figure 6, the system generated 35 explanations from which the SME chose 13 as relevant, and has given an

```
CHOOSE THE EXPLANATIONS
1>  defend-area-mission-D IN company-D-area-of-responsibility
2>  defend-area-mission-D WITH company-D
3>  engagement-area-d PART-OF avenue-of-approach-D
4>  hill-sector-875-2 VISIBLE engagement-area-D
5>  hill-sector-863-2 VISIBLE engagement-area-D
6>  hill-sector-868-1 VISIBLE engagement-area-D
7>  avenue-of-approach-D PART-OF company-D-area-of-responsibility
8>  hill-sector-863-2 VISIBLE mobility-corridor-D PART-OF avenue-of-approach-D
9>  hill-sector-868-1 VISIBLE mobility-corridor-D PART-OF avenue-of-approach-D
10> hill-sector-875-2 IN company-D-area-of-responsibility
11> hill-sector-863-2 IN company-D-area-of-responsibility
12> hill-sector-868-1 IN company-D-area-of-responsibility
13> company-D COMMANDS platoon-D2
14> company-D COMMANDS platoon-D1
15> company-D COMMANDS platoon-D3
16> hill-sector-875-2 QUADRANT 2 AND hill-sector-863-2 QUADRANT 2
18> platoon-D2 EFFECTIVE-RANGE "far" AND platoon-D1 EFFECTIVE-RANGE "far"
...
35> hill-sector-863-2 VISIBLE mobility-corridor-D and hill-sector-868-1 VISIBLE mobility-corridor-D

Enter Selection List: (1 2 3 4 5 6 7 10 11 12 13 14 15)

Give some other explanations [explanation/?/c]:
platoon-D3 WEAPONS-CLASSIFICATION "light"
```

Figure 6: Explanations given for Initial Problem/Solution

IF

plausible upper bound
(SOMETHING AR20)
(SOMETHING AV21 (PART-OF AR20))
(SOMETHING E22 (PART-OF AV21))
(SOMETHING HS-I (IN AR20) (VISIBLE E22))
(SOMETHING HS-TP1(IN AR20) (VISIBLE E22))
(SOMETHING HS-TP2 (IN AR20) (VISIBLE E22))
(SOMETHING C19 (COMMANDS P3)(COMMANDS P2)(COMMANDS P1))
(SOMETHING P1)
(SOMETHING P2)
(SOMETHING P3 (WEAPONS-CLASSIFICATION "light"))
(SOMETHING M23 (WITH C19) (IN AR20))

plausible lower bound
(COMPANY-D-AREA-OF-RESPONSIBILITY AR20)
(COMPANY-D-AVENUE-OF-APPROACH AV21 (PART-OF AR20))
(ENGAGEMENT-AREA-D E22 (PART-OF AR20))
(HILL-SECTOR-868-1 HS-I (IN AR20)(VISIBLE E22))
(HILL-SECTOR-863-2 HS-TP1 (IN AR20) (VISIBLE E22))
(HILL-SECTOR-875-2 HS-TP2 (IN AR20) (VISIBLE E22))
(COMPANY-D C19 (COMMANDS P3)(COMMANDS P2)(COMMANDS P1))
(PLATOON-D1 P1)
(PLATOON-D2 P2)
(PLATOON-D3 P3 (WEAPONS-CLASSIFICATION "light"))
(DEFEND-AREA-MISSION-D M23 (WITH C19) (IN AR20))

THEN
the problem
PLACE-COMPANY C19
IN AR20
TO-DESTROY-ENEMY-IN E22
FOR M23

has the following solution

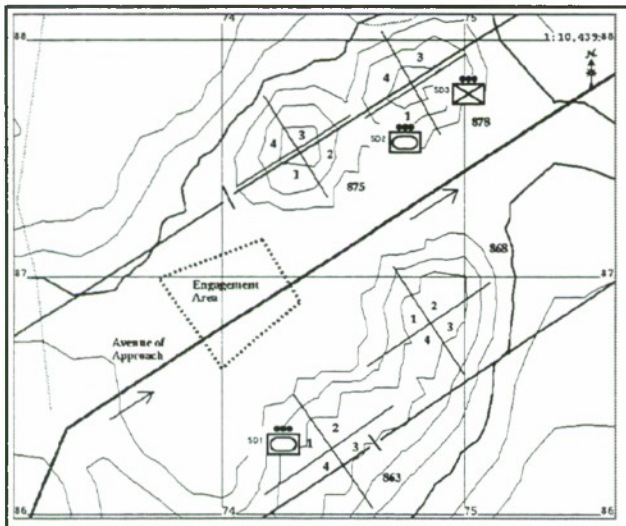
PLACE-INFANTRY-PLATOON	P3 IN HS-I
PLACE-TANK-PLATOON	P1 IN HS-TP1
PLACE-TANK-PLATOON	P2 IN HS-TP2

Figure 7: Initial PVS rule

additional explanation. The chosen explanations indicate that it is important that this is a defend area mission for Company D, that the platoons to be placed belong to Company D, and that these platoons are placed in Company D's area of responsibility, in positions where they can see the engagement area. In the experiments conducted we have found that it is very useful to let the SME control the explanation generation process, so that the SME can balance the effort of choosing explanations from a (possibly long) generated list, with the effort of directly giving explanations.

We stress, however, that while it is important to have some explanations of the initial example, there is no requirement that a complete set of explanations must be specified. Indeed, the assumption made by the agent is that this initial explanation set is incomplete (and possibly even incorrect) and will be completed during experimentation. Consequently, a window with the initial example from Figure 5 will be kept on the screen for the entire duration of the learning session so that the agent can ask additional questions about this example.

The relevant explanations identified by the SME are used by the agent to generate an initial plausible version space for a general placement rule to be learned. This version space is indicated in Figure 7 and will not



Generated Problem:
Place-company company-D
in company-D-area-of-responsibility
to-destroy-enemy-in engagement-area-D
for defend-area-mission

Proposed Solution:
place-infantry-platoon platoon-D3
in hill-sector-878-2
position-tank-platoon platoon-D1
in hill-sector-863-1
position-tank-platoon platoon-D2
in hill-sector-878-1

Could you provide an explanation of the failure?
`hill-sector-878-2 DISTANCE-TO-ENGAGEMENT-AREA "far"`
Could you now provide an explanation of why the initial episode is correct?
`hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close"`

Figure 8: User rejects Company D Placement

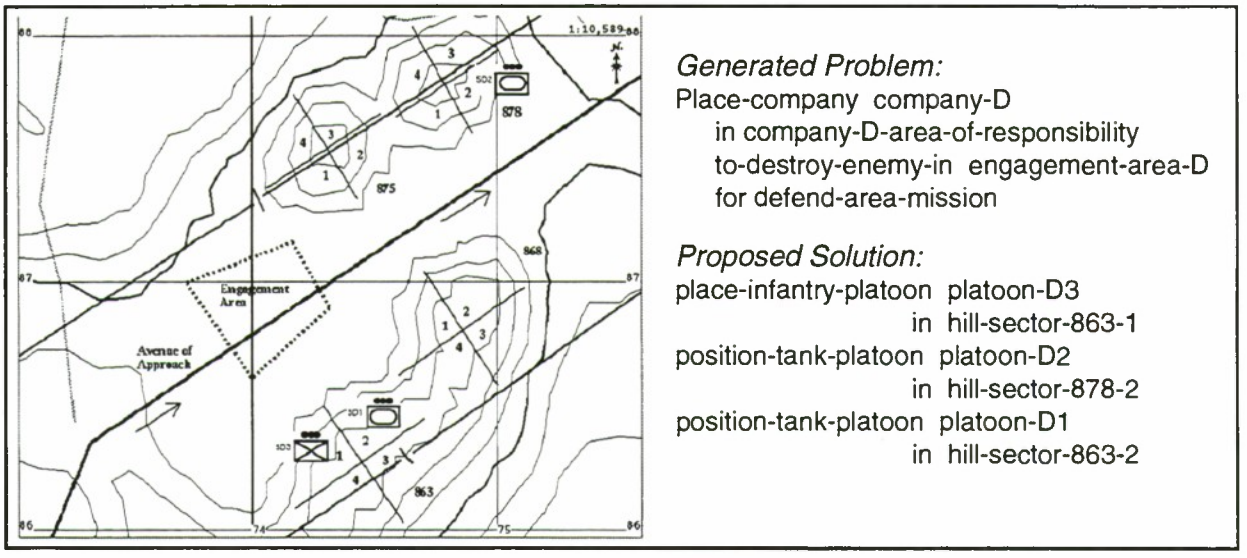


Figure 9: User accepts Company D Placement

be shown to the SME who communicates with the system only through concrete examples and explanations. The conclusion of the rule in Figure 7 is obtained by turning the objects from the initial example (see right hand side of Figure 5) into variables. The plausible lower bound is the conjunction of the selected explanations, reexpressed in terms of the variables from the rule's conclusion. In other words, the plausible lower bound covers only the initial example from Figure 5. The plausible upper bound is an over-generalization of the plausible lower bound, in which individual objects are turned into the most general object "something" and the relationships between the objects are preserved.

The agent will use the plausible version space in Figure 7 to generate other placements for defensive missions, and will show these to the SME, who will accept or reject them. The SME can control this experimentation process by fixing some of the parameters of the defensive mission. For instance, it is useful to ask the agent to initially generate only placements of Company-D in its area of responsibility. This limits the search space the agent must deal with.

The Captain agent generates a new placement of Company-D by simply matching the plausible upper bound of the rule in Figure 7 with the map region in Figure 5. It then proposes the placement to the SME on the ModSAF screen, as shown in the left hand side of Figure 8. The SME rejects this placement and explains that the infantry unit is too far away from the area of engagement.

As a result of these explanations, the property value pair (distance-to-engagement-area "close") is added to the clause for the variable HS-I in both the upper and lower bound:

new plausible upper bound

(SOMETHING HS-I (IN AR20) (VISIBLE E22)
(DISTANCE-TO-ENGAGEMENT-AREA "close")
...

new plausible lower bound

(HILL-SECTOR HS-I (IN AR20) (VISIBLE E22)
(DISTANCE-TO-ENGAGEMENT-AREA "close")
...

Then the agent generates the placement in Figure 9 that is accepted by the user. Consequently, the system makes the following generalizations in the lower bound that correspond to the generalization of the positive examples from Figure 5 and Figure 9:

HILL-SECTOR-868-1 HILL-SECTOR-863-1
→ HILL-SECTOR

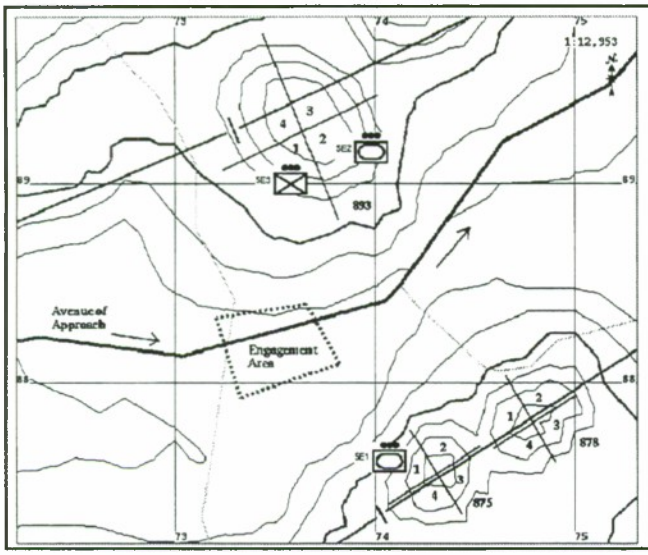
PLATOON-D1 PLATOON-D2 → ARMORED-PLATOON

HILL-SECTOR-863-2 HILL-SECTOR-878-2
→ HILL-SECTOR

HILL-SECTOR-875-2 HILL-SECTOR-863-2
→ HILL-SECTOR

At this point, all the placement examples for Company-D that the system might generate are already covered by the plausible lower bound of the version space. Therefore the SME requires the agent to experiment with placing other companies for defending their areas of responsibility. The system then generates a new example for the SME to validate, as shown in Figure 10. This example comes from a different area on the map, which is in the area of responsibility for Company E.

Because the SME accepted the placement generated by the agent for Company E, the system is able to make a significant reduction in the plausible version space by generalizing the following concepts from the plausible lower bound:



Generated Problem:

Place-company company-E
in company-E-area-of-responsibility
to-destroy-enemy-in engagement-area-E
for defend-area-mission

Proposed Solution:

place-infantry-platoon platoon-E3
in hill-sector-893-1
position-tank-platoon platoon-E2
in hill-sector-893-2
position-tank-platoon platoon-E1
in hill-sector-875-1

Figure 10: Company E Placement generated by Captain

COMPANY-D COMPANY-E → COMPANY

AVENUE-OF-APPROACH-D AVENUE-OF-APPROACH-E
→ AVENUE-OF-APPROACH

ENGAGEMENT-AREA-D ENGAGEMENT-AREA-E
→ ENGAGEMENT-AREA

DEFEND-AREA-MISSION-D DEFEND-AREA-MISSION-E
→ DEFEND-AREA-MISSION

PLATOON-D3 PLATOON-E3 → INFANTRY-PLATOON

COMPANY-D-AREA-OF-RESPONSIBILITY
COMPANY-E-AREA-OF-RESPONSIBILITY
→ AREA-OF-RESPONSIBILITY

After considering one more area of responsibility (for Company F), the agent now learns the rule shown in Figure 4. It is important to stress that while this rule has been learned from five examples, the agent internally examined approximately 5,000 different placements that are covered by the upper bound of the rule in Figure 4. These placements are for the three areas considered so far – for Companies D, E and F. The learning process stopped because the rule was refined to where all the placements that could be generated were covered by the lower bound of the rule being learned (there was no other placement both covered by the plausible upper bound and not covered by the plausible lower bound). It is obviously impractical for a human expert to consider this many solutions individually. However, the SME may, if the SME wishes, continue to verify the learned rule, by examining placements covered by the plausible lower bound.

This illustration gives only a very general outline of the learning method. There are many other kinds of interactions between the SME and the agent. For instance, the SME may explain why a company placement is wrong by pointing to one or more platoons that are not correctly positioned, instead of giving a textual explanation (e.g., in Figure 8 the SME may point to the position of platoon D-3 when asked why the solution was not acceptable). The SME may also choose to give the agent additional examples of good placements. These may cause the generalization of the plausible lower bound or of both the lower and the upper bounds.

During learning, the agent may also accumulate negative or positive exceptions of the rule. These are bad placements that are covered by the plausible lower bound, or good placements which are not covered by the plausible upper bound. In such cases, the agent will attempt to elicit new knowledge (e.g., new features of platoons or their positions that are not defined in the knowledge base) from the SME. These knowledge items will allow the agent to modify the plausible version space of the rule such that the negative and the positive exceptions become negative examples and positive examples, respectively. Some of these knowledge elicitation techniques are described in (Tecuci and Hieb, 1994). Another way of dealing with a rule's exceptions is to split the plausible version space into several plausible version spaces that do not have exceptions. This will, of course, lead to learning more rules for the particular problem.

The general idea of this approach is to allow the SME to teach the agent in a variety of ways, as the SME would train an assistant and to intervene whenever the SME wishes in the teaching process. On the other hand, the agent learner has a very proactive strategy of soliciting explanations in a variety of ways so as to remedy its failures.

Because this approach is based on an expert interacting with, checking and correcting the way the agents solve problems, it produces verified knowledge-based agents.

When the agent has been trained with examples of the typical situations it should be able to cope with, it enters a further phase, *Autonomous Learning*, where it is used in simulations without the assistance of the SME. The training received during the Apprenticeship Learning Phase will allow the agent to solve most of the planning problems through deductive reasoning. However, it will also be able to solve unanticipated problems through plausible reasoning, and to learn from these experiences, in the same way it learned from the SME. The main difference is that the agent must assign credit or blame to the actions by itself. For instance, if the agent generated appropriate orders by using the plausible upper bound condition, it could then generalize the plausible lower bound condition, to cover the respective situation. If, on the other hand, the agent generated incorrect orders, it could need to specialize the rule's conditions. Therefore, the agents developed using this approach will also have the capability of continuously improving themselves during their normal use.

7. Conclusions

In this paper we have presented the Captain tool for building intelligent adaptive agents capable of complex terrain reasoning, which act as automated commanders within the ModSAF distributed interactive simulation environment. In the experiments described, Captain learned placement rules for ModSAF armored company commanders in hilly terrain by generating four to eight examples of company deployments (to show to the SME on the ModSAF graphical interface). The rules were learned from consideration of over 5000 different placements in three different areas of responsibility. Below, we describe some of the benefits obtained from using the Captain approach, describe a few of the future research areas and conclude with a discussion of using Captain to construct agents.

The efficiency of our agent building methodology is achieved through the use of simple plausible version spaces and a human guided heuristic search of these spaces. Plausible version spaces have been inspired by the classical version space concept introduced by Mitchell (1978), developing it along the following directions:

- the ability to learn from only a few examples since the expert's explanations identify the relevant features of the examples. In our experiments we have found that, during a learning session, the system usually needs to generate less than 10 examples, in order to learn a rule that may have several thousands of instances in the knowledge base.
- the ability to learn partially inconsistent rules, when the representation language is incomplete, as well as to guide the elicitation of additional knowledge from the expert, to reduce this incompleteness.
- the use of a heuristic search by limiting the upper and lower bounds to only one conjunctive expression. This avoids a combinatorial explosion of the version space bounds. This might lead to a rule that is not as general as it could be. However, it will always lead to a useful rule that is a generalization of the initial training example provided by the expert.

A characteristic feature of our agent building approach is that an SME trains the agent using a variety of techniques, many of which are similar to how an SME would instruct a human apprentice. This is achieved with the help of ModSAF's graphical user interface that allows the SME to communicate with the agent by placing and moving units on a map. Also, both the SME and the agent actively communicate, with the agent asking questions, and the SME providing answers and training examples. Agent's questions are, in general, easy to answer. Many of them ask for a "yes" or "no" answer (e.g., asking if an example generated by the agent is positive or negative; if some expression is or is not an explanation of some failure; if an object has or does not have a certain feature; if an abduced fact is true or not; etc.). More difficult questions are those asking the SME to provide an explanation of some failure (when the system was not able to propose any) or to indicate the name of a concept covering specific instances (Tecuci & Hieb, 1994). However, in our experiments, we have found that even these questions are not very difficult to answer.

Further research is necessary in four broad areas: improving the basic learning methods that constitute our approach; developing more flexible methods of instruction; developing semantic terrain transformations; and improving the Captain interface to ModSAF.

While the system is able to perform adequately, more work remains to be done on improving efficiency in learning through better search algorithms. We are currently working on simultaneously learning several rules when the incompleteness of the representation language prevents learning one consistent rule. We are also working on elicitation methods for eliminating exceptions as they occur in the learning process.

We are currently working on developing even more flexible methods of instruction that allow the expert

to express whatever instruction is desired at any point in the learning process, as advocated by Huffman (1994) in his work on Instructo-Soar. We are also working on developing additional methods of consistency driven knowledge elicitation, in order to reduce the burden of explanation on the expert.

The terrain reasoning capabilities of Captain are based on semantic terrain transformations, a model transformation process that transforms a digital terrain database into a conceptual semantic network (Hille, et al., 1995). Further work is required to develop and automate this process. In particular, methods for the manipulation and generalization of numbers must be improved, since the current implementation is based on a translation between numeric parameters and symbolic concepts.

We are continuing to enhance Captain's interface to ModSAF. Captain is advancing to the battalion level, where it is learning to establish company area-of-responsibilities and company placements.

The agent building approach illustrated by Captain is based on the Disciple theory and methodology that integrates multistrategy machine learning (Michalski and Tecuci, 1994) and knowledge acquisition (Tecuci & Kodratoff, 1995), within the framework of apprenticeship learning (Mitchell et al., 1985; Tecuci, 1988; Wilkins, 1990). It synergistically combines explanation-based learning, learning by analogy, empirical inductive learning from examples, conceptual clustering, and learning by instruction.

Building on the work on intelligent agents (Laird and Rosenbloom 1990; Gordon and Subramanian 1993; Van de Velde 1993; Maes, 1994; Mitchell et al., 1994), we are also working on integrating experience-based learning with the available learning strategies, to be used when the agent is acting autonomously.

Captain's learning approach offers several benefits to CGF developers. It produces verified knowledge bases for command agents and, when new knowledge is added or new behaviors learned, the system will verify that the existing behaviors are still correct. When a behavior needs to be modified, the same rules can be easily adjusted through the "apprenticeship" process, rather than thrown away and rewritten. These capabilities make Captain appropriate for use in large exercises, where many units need to be quickly modified to a new doctrine. Most importantly, in Captain the role of the knowledge engineer is significantly reduced and the learning process is interactive – producing better deliberative behavior with less time than is needed using traditional knowledge acquisition methods.

8. Acknowledgment

This research was conducted in the Computer Science Department and the Center for Excellence in Command, Control, Communications & Intelligence at George Mason University. Work on ModSAF applications was sponsored in part by the Defense Modeling and Simulation Office under contract DCA100-91-C-0033.

9. References

Ceranowicz A., (1994). ModSAF Capabilities. *4th Conference on Computer Generated Forces and Behavior Representation*, May, Orlando, Florida.

Gammack J.G. (1987). Different Techniques and Different Aspects on Declarative Knowledge. In A.L. Kidd, Ed. *Knowledge Acquisition for Expert Systems: A Practical Handbook*, Plenum Press.

Gordon, D. & Subramanian, D. (1993). A Multistrategy Learning Scheme for Agent Knowledge Acquisition. *Informatica*, 17(4).

Hieb, M.R., Hille D. and Tecuci, G. 1993. Designing a Computer Opponent for War Games: Integrating Planning, Learning and Knowledge Acquisition in WARGLES. In *Proceedings of the 1993 AAAI Fall Symposium on Games: Learning and Planning*, AAAI Press Technical Report FS-93-02, Menlo Park, CA.

Hille D., Hieb M.R., Pullen J.M. & Tecuci G. (1995). Abstracting Terrain Data through Semantic Terrain Transformations. *5th Conference on Computer Generated Forces and Behavioral Representation*. Orlando, Florida.

Hille D., Hieb, M.R. & Tecuci, G. 1994. Captain: Building Agents that Plan and Learn. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*.

Huffman, S.B. (1994). Instructable Autonomous Agents. *PhD Thesis*. Department of Computer Science and Engineering. University of Michigan.

Laird J.E. & Rosenbloom P.S. (1990). Integrating Execution Planning and Learning in Soar for External Environments, *Proceedings. AAAI-90*. Boston.

Maes, P., (1994). Agents That Reduce Work and Information Overload, in *Communications of the ACM*, 37(7).

Michalski R.S. & Tecuci G. Eds. (1994). *Machine Learning: A Multistrategy Approach*, Vol. IV, Morgan Kaufmann, San Mateo.

Mitchell, T.M. (1978). *Version Spaces: An Approach to Concept Learning*, Doctoral Dissertation, Stanford University.

Mitchell T. M., Mahadevan S. & Steinberg L. 1., (1985). LEAP: A Learning Apprentice System

for VLSI Design, in *Proceedings of the IJCAI-85,* Los Angeles, Morgan Kaufmann.

Mitchell T. M., Caruana R., Freitag D., McDermott J. & Zabowski D., (1994). Experience with a Learning Personal Assistant, in *Communications of the ACM,* 37:7.

Pullen, J.M. (1994). Networking for Distributed Virtual Simulation. In B. Plattner & J. Kiers Eds, *Proceedings. of INET'94/JENC5.* Internet Society (isoc@isoc.org).

Tecuci, G. (1988). DISCIPLE: A Theory, Methodology and System for Learning Expert Knowledge, Ph.D. Thesis, University of Paris South.

Tecuci G. (1992). "Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base," *IEEE Transactions of SMC.* 22(6).

Tecuci, G. & Hieb, M.R. (1994). Consistency-driven Knowledge Elicitation: Using a Machine Learning-oriented Knowledge Representation to Integrate Learning and Knowledge Elicitation in NeoDISCIPLE. *Knowledge Acquisition,* 6(1).

Tecuci, G., Hieb M.R., Hille D. & Pullen J.M. (1994). Building Adaptive Autonomous Agents for Adversarial Domains, *Proceedings of the AAAI 94 Fall Symposium – Planning and Learning: On To Real Applications.*

Tecuci G. & Kodratoff Y. (Eds). (1995). *Machine Learning and Knowledge Acquisition: Integrated Approaches*, Academic Press.

Van de Velde W., (Ed). (1993). *Towards Learning Robots.* MIT Press: Cambridge, MA.

Wilkins D.C., (1990). Knowledge Base Refinement as Improving an Incorrect and Incomplete Domain Theory, in Y. Kodratoff and R. S. Michalski (Eds). *Machine Learning: An Artificial Intelligence Approach.* Vol. 3. Morgan Kaufmann.

10. Authors' Biographies

Michael Hieb is a PhD candidate in Information Technology at George Mason University in Virginia. He is currently a researcher at the Computer Science Department of George Mason University working on automated knowledge acquisition of behavior in complex domains, such as ModSAF. He is also researching interaction modes for knowledge acquisition interfaces. He has published papers in the areas of knowledge acquisition, multistrategy learning, and plausible reasoning. He has served as a consultant in AI for CSC and implemented a distributed problem solving testbed at the Goddard Space Flight Center for IntelliTek, Inc.

Dr. Gheorghe Tecuci is Associate Professor of Computer Science at George Mason University. He has published over 70 scientific papers, mostly in the area of artificial intelligence. Gheorghe Tecuci is a member of the Romanian Academy and is known for his pioneering work on multistrategy machine learning and its integration with knowledge acquisition. He developed Disciple, which is one of the first multistrategy learning systems, and co-edited the first books on multistrategy learning and on the integration of machine learning and knowledge acquisition. He was the program chairman of the first workshops in these areas (MSL-91, MSL-93, IJCAI-93: ML & KA).

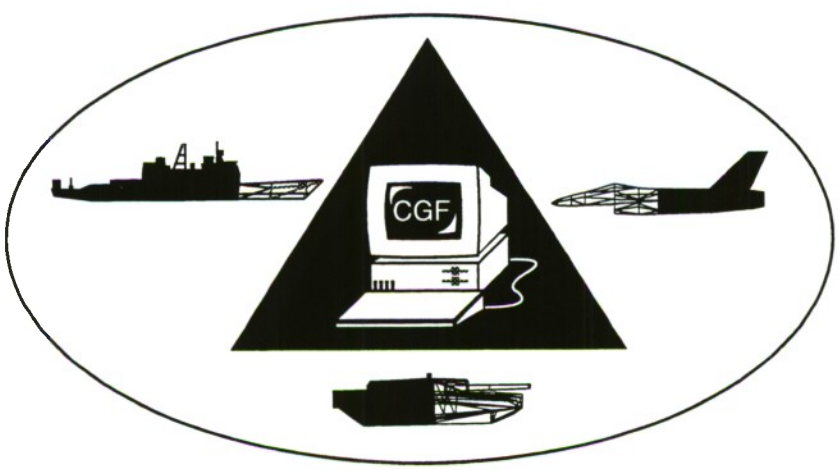
Dr. Mark Pullen is Associate Professor of Computer Science at George Mason University. He also has an appointment with the Center for Excellence in Command, Control, Communications and Intelligence. Dr. Mark Pullen was employed by the Defense Advanced Research Projects Agency (DARPA) from 1986 to 1992, where he was Program Manager for Advanced Computing, Networking and Distributed Simulation, and Deputy Director of the Tactical Technology Office and the Information Science and Technology Office. His research interests include distributed and parallel computing systems and their applications to educational and military simulations.

Dr. Andrew Ceranowicz is the manager of the Semi-Automated Forces group at LADS. He has been working on Distributed Interactive Simulation and Semi-Automated Forces since 1986 when he joined BBN to work on the ARPA SIMNET Project. Since then he has contributed to the ARPA ODIN Project for intelligence visualization, battle reenactment, and ModSAF. Prior to his work at Loral, Dr. Ceranowicz was a member of the technical staff at Draper Laboratory working on expert systems and GPS applications. Dr. Ceranowicz earned his Ph.D. in control theory from the Ohio State University.

David Hille is a computer scientist at ANSER, a public service research institute. He received a Masters of Science in Computer Science at Syracuse University and is currently a Doctor of Science candidate in the Information Technology PhD program at George Mason University. He has designed military simulations published commercially by Strategic Simulations, Inc. and helped to develop the wargame methodology for technology base seminar wargames. He has been working on the Captain Learning and Planning System project, a system that performs as an automated agent in simulations.

Session 4a: Command & Control Modeling I

Page, UK DRA
Karr, UCF/IST
Lankester, UK DRA


CGF

An Automated CBS OPFOR

Ian Page
Simulation, Training & AI Research Group
LSC1, Building Q27
Defence Research Agency
Fort Halstead, Sevenoaks, Kent, TN14 7BP
United Kingdom
ipage@dra.hmg.gb

Gary Kendall
Logica UK Ltd
Stephenson House
75 Hampstead Road
London, NW1 2PL
United Kingdom
garyk@logcam.co.uk

1. Abstract

Training is vital to sustain a combat ready state. As the availability of real terrain areas to train on decreases, the importance of simulation increases.

Simulations for Command and Staff Training usually require a large number of controllers to operate them. One of the controllers' tasks is to make command decisions for units not under player control. It is a research aim of the UK Ministry of Defence to investigate the role of intelligent Computer Generated Forces to support controllers with this task.

This paper examines the issues involved with increased controller automation using the Corps Battle Simulation (CBS). Specifically it describes a proof of principle demonstrator showing the complete automation of controller functions for both OPFOR and Blue forces. This is achieved using rule based command agents within the existing GEKNOFLEXE model of battlefield decision making.

The demonstrator controls approximately 200 units over the course of a fifteen hour battle, with over 5,000 orders submitted to CBS. A task that would require approximately ten controllers.

Current research is concentrating on the UK Army's recently installed Higher Formation Trainer - ABACUS.

2. Introduction

2.1 Requirement

Maintaining an acceptable level of combat readiness for the armed forces is extremely important, and increasingly difficult, with today's rapidly changing world political scene. Key to sustaining this readiness is training. Unfortunately, due to a combination of environmental pressures, decreasing defence budgets and increasing weapon capabilities, the availability of training areas is diminishing.

Military commanders are therefore looking to simulation technology to help address this shortfall. Within the area of Command and Staff Training (CAST), there is a requirement for simulations to run Command Post Exercises (CPXs). During a CPX, particular headquarters are tested. To increase realism, many superior, subordinate and flanking forces, and their headquarters, are also required and have therefore to be simulated.

To play these roles, human controllers are normally required to make the command decisions for those units in the simulation which have no corresponding real world players. This includes **all** of the opposition forces (OPFOR). For a large exercise, the total number of controllers can be in the hundreds. These people are expensive, difficult to obtain and receive little, or no, training benefit from the exercise.

What current operational simulation technology is not yet able to offer is the automated decision making capability typical of such command headquarters. Research is therefore being conducted to investigate the role of so-called intelligent Computer Generated Force (iCGF) technologies to increase automation of the decision making process.

2.2 Aims

The main aims of the research presented here were to investigate the feasibility of connecting the GEKNOFLEXE studies tool, which includes an iCGF system (Lankester & Robinson, 1994), to a CAST simulation, namely the Corps Battle Simulation (CBS). The purpose of this was to gain a better understanding of the issues involved in automating CAST controller functions.

CBS (JPL, 1991) is a constructive, aggregate level battlefield simulation designed to run Corps and Divisional exercises. CBS takes in orders from the players, via human controllers. The effects of these are modelled in CBS and the results fed back to the players,

again via the controllers.

GEKNOFLEXE is a fully automated, two-sided model of battlefield C^3I (Command, Control, Communications and Intelligence). It models the decision making process of command headquarters using knowledge based software constructs known as 'command agents'. The GEKNOFLEXE command agents form a distinct group of objects separate from the GEKNOFLEXE battlefield simulation which handles events like sighting, movement and attrition. Command agents make the same decisions which CAST simulations presently require human controllers to make. By connecting GEKNOFLEXE command agents to CBS, automation of controller functions should be possible.

Feasibility, design, implementation and validation stages for this connection were to be completed during a twelve month period. The initial stages of which have already been presented (Cox, Gibb & Page, 1994). This paper describes the results from this work.

A number of issues need to be resolved to produce a satisfactory level of controller automation with command agents. These are discussed by Cox, et al. (1994), but generally cover the need for realistic behaviours over a wide range of settings, a sufficient representation of unit types, manual override facilities and mechanisms for validation. It was important, therefore, to scope the research down to the immediate problem of connecting the GEKNOFLEXE iCGF to the CBS CAST system. To achieve this, it was intended to simplify the approach wherever possible.

Consequently, the feasibility study recommended that the CBS OPFOR be fully automated by GEKNOFLEXE. This has several advantages.

Fully automating the OPFOR removes the immediate need to address the complex issue of interactions between players, controllers and the iCGF. For example, for players to communicate directly with the iCGF would require speech generation and synthesis well outside the scope of this research. Allowing the iCGF to function entirely under its own control also avoids the need for controller iCGF supervisory facilities.

CBS has a simpler OPFOR representation of logistic, maintenance and medical functions than for own forces. Since GEKNOFLEXE had no model of these anyway, it was more sensible to let GEKNOFLEXE control CBS OPFOR units.

Thus it was the intention to produce a working demonstrator by the end of this research which would show full automation of the OPFOR units within CBS. This would then meet the main aim of investigating the feasibility of the GEKNOFLEXE iCGF acting as an automated CBS controller. By doing this, a more comprehensive understanding of controller automation in general would be attained.

3. Design and Implementation

3.1 Ideal System Architecture

The first stage of the design was to plan an ideal system architecture for automating the CBS OPFOR which has already been discussed by Cox, et al. (1994). From this ideal architecture a more practical design could be derived, if necessary.

Figure 1 shows a typical CBS configuration for a headquarters under exercise. The trainee staff communicate with their controllers as normal. Other controllers play the roles of superior, subordinate and flanking forces present within the setting, for whom there are no player counterparts.

The controllers therefore input orders into their CBS workstations, and report back events occurring on the battlefield to the trainees. The trainees should not be aware of the underlying computer system controlling battlefield events. To all intents and purposes, the trainees are in a real battle situation, communicating with other real forces. In reality these are all modelled within CBS under controller command.

A senior controller monitors the exercise, and makes adjustments to the simulation as and when necessary, to maintain the exercise objectives.

OPFOR units have no OPFOR trainees. The OPFOR is therefore fully commanded by the OPFOR controllers.

Figure 1 shows GEKNOFLEXE OPFOR command agents in place of human controllers. Whereas human controllers input orders manually into their CBS workstation, and receive reports from CBS on their workstation, GEKNOFLEXE command agents are electronically connected to CBS via its generic interface (GI). CBS has a GI to allow the majority of controller functions to be electronically input, and CBS reports to be output, from and to other software.

3.2 Proposed System Architecture

Based on this ideal design, a proposed system architecture was constructed (see figure 2). The main constraints preventing the implementation of the ideal

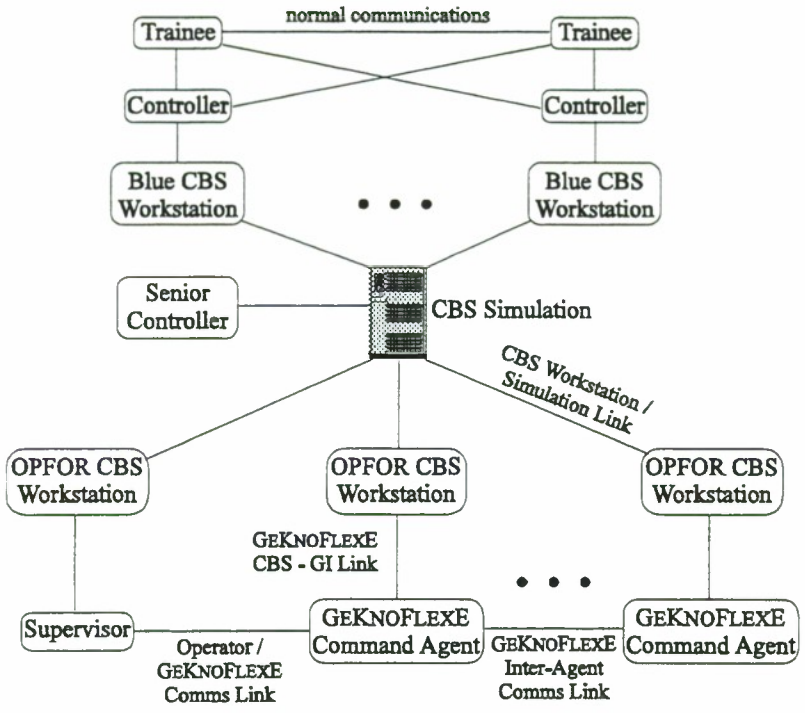


Figure 1: Ideal system architecture

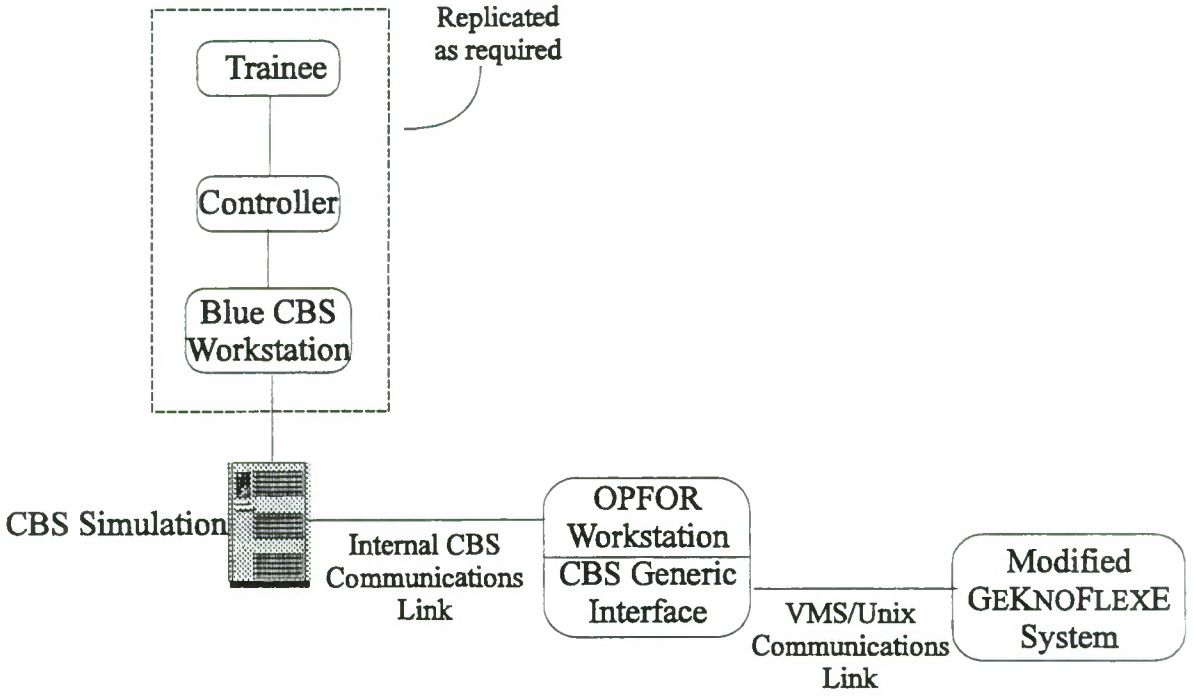


Figure 2: Proposed system architecture

design, and influencing the proposed architecture are described below.

The initial research programme was only to last for a twelve month period, for financial reasons. The UK Army was also installing its own Higher Formation Trainer (HFT) - ABACUS (All Arms Battlefield Computer Simulation). After twelve months would be a suitable breakpoint in the long term research programme to review how the work was progressing. This would allow the lessons learnt to date to be assessed and the research focus to switch to ABACUS, reflecting the CAST needs of the UK Army.

It was also desirable to make as few, if any, changes to the still developing GEKNOFLEXE system and CBS, as an operational trainer, could not be modified at all. It was essential, therefore, to make any link between these two systems as independent as possible from the actual systems themselves.

Figure 2 shows the proposed design and figure 3 the processes involved in automating the OPFOR. Although the Blue situation remains unchanged from the ideal design, the OPFOR is subtly different, as explained below.

Instead of separating out the GEKNOFLEXE command agents from the GEKNOFLEXE simulation, it was decided to modify the GEKNOFLEXE simulation so that it would emulate CBS. This GEKNOFLEXE 'simulation emulation' would receive battlefield event updates from CBS, via the GI, and reflect these changes in the GEKNOFLEXE simulation. In effect, control of the GEKNOFLEXE simulation would be handed over to CBS. In this way, GEKNOFLEXE would not need to be altered as much as if the command agents were isolated from the GEKNOFLEXE simulation.

The GEKNOFLEXE simulation does not totally emulate CBS. In the case of its sightings model, GEKNOFLEXE has this integrated into the simulation and includes facilities for modelling RPVs (Remotely Piloted Vehicles). This capability was not directly available from the version of CBS used. So sightings reports received by the command agents are taken from the GEKNOFLEXE simulation, rather than CBS.

Limiting the OPFOR representation is the battlefield functionality of GEKNOFLEXE. The version of GEKNOFLEXE used was only able to reason about armoured, armoured infantry, artillery, mortar, recce and command headquarter units. Division, Brigade and

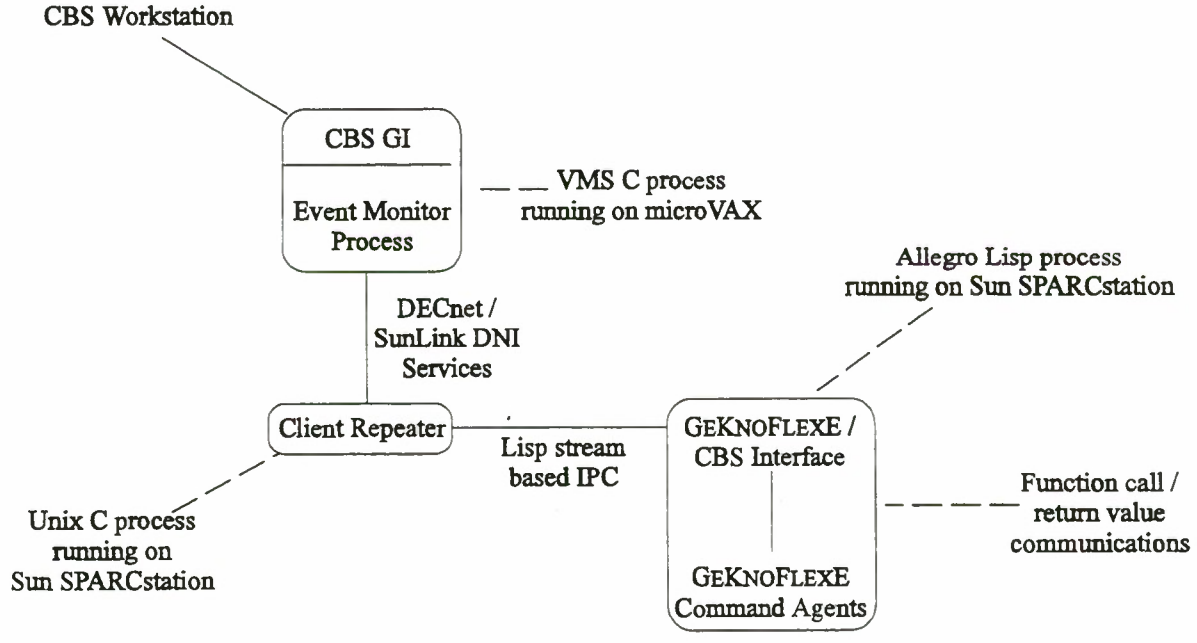


Figure 3: Detailed process structure

Battalion level battlefield decision making is modelled, with Company units simply obeying orders.

This version uses a post-CFE (Conventional Forces in Europe) setting, which has an OPFOR tank Division attacking from the east, over the former East Germany, against a Blue force based on the ACE (Allied Command Europe) Rapid Reaction Corps fighting a covering battle. It was also necessary to configure the CBS scenario to match this GEKNOFLEXE setting.

As already stated, GEKNOFLEXE is a developing system and its latest version is already much enhanced over the version taken for this work. This new version considerably increases the military functionality and terrain resolution modelled, and uses an intervention-based scenario.

Figure 3 depicts the processes created to achieve OPFOR controller automation. The event monitor process runs on the MicroVAX hosting the CBS GI. This process monitors CBS for any update notifications, e.g. a change to a unit's location, and forwards this information to the repeater process on the Sun SPARCstation which hosts GEKNOFLEXE. The other function of the event monitor process is to transmit orders originating from GEKNOFLEXE to CBS using the GI.

The client repeater process performs no processing itself, but rather acts as a buffer between the modified GEKNOFLEXE process and the event monitor process. The client repeater process is necessary because it is difficult for the other two processes to communicate directly. The client repeater talks to the event monitor via the DECnet communications protocol and the client repeater communicates with the modified GEKNOFLEXE process through a Unix pipe.

The modified GEKNOFLEXE process is hosted on a Sun SPARCstation and listens for information coming from the client repeater process, as well as passing GEKNOFLEXE command agent orders to the client repeater process. When a report from CBS arrives, a layer of software within the GEKNOFLEXE Lisp process updates the GEKNOFLEXE simulation, thus keeping both the GEKNOFLEXE and CBS simulations synchronised. This same software also routes GEKNOFLEXE orders to CBS, via the various processes.

In this way, GEKNOFLEXE command agent decisions, in the form of orders, are passed to CBS. Similarly, event updates are routed from CBS to the GEKNOFLEXE simulation for the GEKNOFLEXE command agents to act upon.

A final consequence of producing a fully automated OPFOR, and ignoring the provision of supervisory facilities for controllers, is that the role of the senior controller is dramatically reduced. This is because the GEKNOFLEXE command agents are not able to cope with the non-real world activities, e.g. magic moves and over-ruling of decisions, often associated with the senior controller function.

3.3 Implementation of the Proposed System

The implementation largely followed the design specified by the proposed system architecture. The majority of the effort concentrated on building the link to the GEKNOFLEXE simulation. The implemented CBS simulation emulation allowed the iCGF components of GEKNOFLEXE to retain their existing software harness. Though not an elegant solution as it requires two simulations (GEKNOFLEXE and CBS) to be kept synchronised, it allowed a proof of principle demonstration of the automation of controller functions to be produced in a remarkably short time. It took approximately twelve man months of effort over a six month period.

This resulted in a demonstration system showing the complete automation of both OPFOR and Blue CBS forces. Control of the Blue forces can also be handed over to human controllers, thus allowing players to fight an automated enemy. Further details of the demonstration system are given in §4.2. A description of the problems faced while implementing the demonstrator system are presented below.

3.4 Problems

3.4.1 Restrictions

As mentioned earlier, GEKNOFLEXE has a limited battlefield representation compared to the functionality offered by CBS, and this introduced a number of restrictions to the demonstration scenario.

Foremost of these is the ORBAT. It was necessary to configure the CBS ORBAT to match that of GEKNOFLEXE. This emphasised resolution problems arising from the different levels at which each system was designed to operate. For instance, CBS typically represents recce platoons, whereas GEKNOFLEXE models recce sections.

GEKNOFLEXE allows opposing forces to occupy the same terrain cell, however, the version of CBS used does not. In the GEKNOFLEXE simulation, units have a probabilistic chance of seeing each other. It is not uncommon for opposing recce sections to occupy the

same 3km terrain cell as the chances of them spotting each other are relatively low. Hence, such small units often pass by one another unsighted. This cannot happen in CBS, which was not designed to represent units in such detail.

It was therefore necessary to implement a mechanism to halt a unit before it entered an enemy occupied terrain cell. If the unit is not engaged, then after an appropriate delay it is 'magic moved' to where it would have been had it passed conventionally through the enemy cell.

Similarly, a small unit like a recce section can hold up a significantly larger enemy force for longer than would be expected, again, because opposing forces are not allowed in the same terrain cell. The effects of this can be reduced by forcing such small units to operate under the CBS 'avoid combat' mode. This reduces the chances of such units engaging.

These problems arise purely due to our use of CBS at levels for which CBS was not designed.

Since GEKNOFLEXE had no representation of logistics, it was necessary to give all units infinite supplies. This is justifiable as the GEKNOFLEXE command agents, in this post-CFE scenario, have been validated for only a fifteen hour battle period and do not abuse their limitless supplies.

3.4.2 GEKNOFLEXE Command Agent Dependencies

Because the GEKNOFLEXE command agents have been developed with their own simulation, a couple of command agent dependencies on it were uncovered.

GEKNOFLEXE has a fairly simple congestion model, and only three units, moving in column formation, are allowed in any one terrain cell. When facing an enemy or obstacle, the GEKNOFLEXE congestion model halts the leading three units and consequently all the following units also halt. However, CBS does allow more than three units into the same terrain cell. This means that when units halt, units following the leading three will not necessarily stop. To solve this, a basic congestion model was implemented to cause enough extra congestion to make CBS halt any following units. In the latest version of GEKNOFLEXE, command agents control all aspects of unit movement.

When replanning routes, instead of GEKNOFLEXE command agents sending reroute orders to units, the low level route within the GEKNOFLEXE simulation is changed. As routes are much simpler in CBS, a change to the appropriate GEKNOFLEXE knowledge base was implemented to make the command agents also send new movement orders down to affected units.

3.4.3 Command Agent Flexibility

One of the initial fears was that the command agents would not appear realistic when operating within the CBS environment, since they were not designed to work outside of the GEKNOFLEXE simulation. However, the command agents were found to be remarkably robust and, with the exception of the few cases described below, act in a reasonably believable manner.

When replanning routes, sometimes a route is chosen which, for example, crosses a river instead of using a suitable nearby bridge. This is due to the reroute algorithms working under the constraints of avoiding enemy occupied terrain cells, where these cells are 3km across. Nearby bridges are ignored if they are in an enemy occupied cell, even though in reality the bridge may be only a few hundred metres from the unit and over 2km from the enemy unit.

When a column formation is disrupted, the command agents are not always able to reform the column in the correct order.

Command agents frequently send fire support orders to artillery units for targets outside their range. These orders have to be filtered out before being sent to CBS.

3.4.4 Implementation Problems

During implementation, a number of unexpected problems arose. Chief among these related to the communications link between the GEKNOFLEXE Lisp process and the CBS GI. These mainly concerned limitations in the development environments used and are of little relevance to the research conducted.

Another problem concerned the apparent failure of CBS to report all unit updates to the GI. This is particularly the case when a number of changes to a unit occur in quick succession. For example, if an artillery unit fires only a few rounds, the following update to say the unit has ceased firing may not be sent. This leaves the GEKNOFLEXE command agents thinking the unit is still firing and so will not task that unit with other fire orders. The problem only affects a small number of the units and so has a limited effect on the simulation.

3.4.5 Simulation Differences

In addition to the already mentioned problem of differing levels of resolution between the CBS and GEKNOFLEXE simulations, a number of other differences were observed.

When converting between the GEKNOFLEXE simulation

and CBS coordinates, only a 200-300 metre accuracy level could be achieved. This led to particular problems when a route along a road is given. This can translate to CBS as a series of points near a road, and consequently a lot of unnecessary off-road movements could be observed.

Only twenty points could be specified in a CBS movement order. Command agents were not so restricted. So a low level monitoring process had to be implemented in the GEKNOFLEXE simulation emulation to issue new movement orders to CBS as and when required.

4. Achievements

4.1 Lessons Learnt

The main aims of this research were to gain a better understanding of the issues involved in automating CAST controller functions. This was achieved by designing and implementing a proof of principle research prototype demonstrating the automation of OPFOR controller functions in CBS. A number of lessons were learnt during this process.

4.1.1 Design

The importance of conducting a thorough feasibility study and using this to produce a well thought out design cannot be emphasised enough. These initial stages, while not a guarantee to success, help to drastically reduce the problems encountered during the implementation stage.

4.1.2 Software & Documentation

The extensive documentation set and the reliability of the CBS software meant that relatively few problems were encountered with CBS. Similarly, few problems with GEKNOFLEXE were observed, again due to the robustness of the software. The lesson to be learnt here is that well documented and reliable software significantly eases the connection process of distinct systems.

4.1.3 Command Agent Assumptions

In relation to congestion (see §3.4.2), a more realistic method of halting a column needs to be implemented. Merely halting the leading three units causes problems outside the GEKNOFLEXE simulation and this has been addressed in the latest version of GEKNOFLEXE.

Regarding movement, the GEKNOFLEXE simulation determines if a moving unit has reached its destination by testing whether the current location matches the end route location. Although CBS end movement reports match those in the movement order, this may not be the case with other simulations. This is an example of the risk of using two synchronised simulations.

4.1.4 Simulation Synchronisation

The adopted approach of having a GEKNOFLEXE simulation emulation leads to obvious problems in keeping both the GEKNOFLEXE and CBS simulations in step with each other. The reason for this double simulation approach was to allow a proof of principle demonstrator to be quickly implemented. This should not be viewed as a long term solution to the problem of controller automation. The overheads of having two simulations in a real training exercise would soon become prohibitive.

4.1.5 Simulation Resolution

CBS and GEKNOFLEXE are different systems designed to do different tasks. What is acceptable at a coarse resolution in CBS is not necessarily adequate in GEKNOFLEXE, e.g. opposing units not being allowed in the same terrain cell. What has been learnt here is that solutions can generally be found, which although by no means optimal, do work. These problems would be reduced if both systems were designed to function at the same resolution, as is more the case with GEKNOFLEXE command agents and ABACUS.

4.1.6 Command Agent Capabilities

One of the aims of this piece of research was to determine whether or not the GEKNOFLEXE command agents were capable of providing an OPFOR in a training environment. The command agents were realistically able to run the full fifteen hour scenario whilst connected to CBS without any changes to the existing knowledge base rules. When fighting against a totally free play, human controlled Blue force for eight hours (see §4.2), the Blue players acknowledged that they had fought a realistic opposition, given the constraints of the scenario. This success indicates the immense potential of command agents for controller automation.

4.1.7 Command Agent Inabilities

GEKNOFLEXE command agents need to be able to control engineering functions, reason more realistically about reforming a disrupted column, and check the range of artillery targets before issuing impossible artillery orders. These are specific problems which are already being addressed as part of new GEKNOFLEXE releases.

What needs greater understanding is the production of a wider variety of command agents which can work in a large number of different scenarios. Current

expectations are that these are feasible, but very costly to develop.

Command agents are also unable to make realistic decisions based on real world information alone. Taking the example of data fusion, software techniques are unable to replicate the human information fusion process. So GEKNOFLEXE command agents 'cheat', and are provided with sufficient ground truth information via so-called 'back door' mechanisms to the simulation, to allow realistic decisions to be made. Such techniques are valid, however, so long as they do not detract from training effectiveness.

4.1.8 Validation

GEKNOFLEXE itself has been validated by independent military experts to confirm that it provides a credible military engagement between two opposing forces, within the context of the scenario and forces modelled. Validation of the CBS automated force behaviour remains somewhat subjective, since validation is context dependent. What is valid in one simulation environment, e.g. the GEKNOFLEXE simulation, is not necessarily valid in another, for instance CBS. Differences in the CBS battlefield simulation, and variation caused by human controlled Blue forces, cause the OPFOR command agents to react in different ways. As a result, new combinations of decisions are produced which have not been previously validated in GEKNOFLEXE. Nevertheless, in a limited trial exercise (see §4.2), the GEKNOFLEXE command agents proved to be robust and provided a reasonable OPFOR.

4.1.9 Coordinate Systems

Whereas using a lat/long coordinate system is more accurate for large areas, army users prefer to see UTM (Universal Transverse Mercator) locations. A translation process was required to convert from GEKNOFLEXE coordinates to the UTM coordinates used by CBS. Though it was only necessary to implement conversions for one UTM zone.

4.2 Demonstration System

The demonstration system is the culmination of this research work. It illustrates the capability for the practical automation of controller functions. Although the original aim was to concentrate on purely an automated OPFOR, the system actually provides for automation of both an aggressive OPFOR behaviour and a fighting withdrawal behaviour by Blue forces. The system is able to operate in one of three modes:

- both Blue and OPFOR fully automated by the GEKNOFLEXE command agents. This is useful when demonstrating or debugging the system, as controllers are not required.

- as above, but with a once-only handover of Blue force control to human controllers, thus disabling the Blue GEKNOFLEXE command agents from further play. This allows the scenario to be automatically advanced to an appropriate point in the battle.

- GEKNOFLEXE command agents playing the OPFOR, human controllers supervising the Blue forces.

While playing the scenario, GEKNOFLEXE is in control of approximately 140 OPFOR units and about 60 Blue units. During the course of a fifteen hour scenario, a total of around 5,000 orders are submitted to CBS from the GEKNOFLEXE command agents. If we assume a human controller takes about two minutes to issue a particular order, then it would require over ten controllers issuing orders non-stop for fifteen hours real time to issue 5,000 orders.

A 'mini-exercise' to test out the demonstration system was conducted. This consisted of two military personnel (a serving Major and a retired Colonel) controlling the Blue forces and fighting a fully automated opposition over a period of eight hours. Given the constraints of the scenario, they reported that they had met a credible force.

5. Future Research

The research has now reached a natural breakpoint. A proof of principle system, showing the automation of CBS controller functions by GEKNOFLEXE command agents, has been successfully demonstrated. During this time, the UK Army's HFT ABACUS has been installed and partially accepted. This now forms the natural focus for further research into the automation of controller functions.

The wide applicability of the GEKNOFLEXE command agents to areas outside their originally intended use, i.e. C^3I studies, has been largely acknowledged and the Command Agents Research (CARE) programme instigated (Lankester, 1995). The main aims of this involve the separation of the GEKNOFLEXE decision making components (command agents) from the GEKNOFLEXE simulation, and the production of further command agents. These will be available for use by the research programmes funding CARE.

For the next stage of controller automation research, it seems sensible to consider the implications of connecting ABACUS with the command agents

emanating from the CARE programme. A number of issues need to be tackled, and these are briefly outlined below.

A connection mechanism to ABACUS needs to be developed. ABACUS does not have a GI like CBS, and so a library of routines need to be developed to provide a sufficient level of functionality for command agents to control ABACUS units.

Communications between players, controllers and command agents need improving. Facilities are required for exercise controllers to monitor, understand and alter command agent activities. This requires a Graphical User Interface (GUI) to readily present the information needed by controllers to allow them to comprehend the what and why of command agent decision making. The capability to then modify command agent states is necessary should a decision need to be overruled or the state of the exercise changed.

Actual verbal communication between players and command agents would require well developed speech recognition and generation software, beyond the current levels of such technology.

Command agent to command agent communications needs to be examined if command agents are to work within a distributed framework such as the Distributed Interactive Simulation (DIS) network. This requires an open, modular architecture. The ARPA (Advanced Research Projects Agency) led Command and Control Simulation Interface Language (CCSIL) may well prove useful for such communications, and further investigation is warranted. There are concerns, however, with CCSIL's philosophy of only providing real world information. It is currently believed that command agents need more than just real world information to model command decision making.

Training exercises can now take place in a wide range of locations, under various concepts of operation, and with diverse ORBATs. Command agents need to be flexible enough to cope. This requires significant effort to optimise the process by which knowledge is captured and encapsulated in command agent form.

The validity of command agent behaviours within a CAST environment needs to be further assessed. It is vital that units controlled by command agents, both OPFOR and friendly, act in a realistic manner so that the trainees are presented with a credible setting. There is much uncertainty within the community as to how to best validate force behaviours, particularly in new scenarios.

6. Conclusions

This research has shown that the automation of controller functions is viable. The demonstrator system has shown itself to be an excellent means of both understanding the issues involved with the automation of CAST controller functions and illustrating the research objectives.

The practical lessons learnt will provide a sound basis for future research into automating ABACUS controller functions. The main lessons learnt relate to the provision of a well defined interface between simulations and iCGFs, the increased functionality of command agents, and the need to avoid having more than one simulation in the system.

There are a number of risks associated with the plans for future research. Dependencies on ABACUS and CARE are high. ABACUS is a new system about which very little is known or understood by the research team. However, the ABACUS Higher Formation Trainer (HFT) best reflects the UK Army's requirements from a CAST. CARE is itself a research programme and consequently comes with a certain element of risk. But no other software appears to offer the advanced decision making capability provided by GEKNOFLEXE command agents.

The cost benefits from reducing the numbers of controllers required to run a CAST system are obvious. GEKNOFLEXE command agents show the potential to achieve this. Benefits beyond financial savings include the increased usage of CAST systems since they will become more affordable to run. Controllers, already in limited supply, will be able to focus on the important aspects of an exercise, thus improving the training benefits to the players. Increasing the automation could ultimately provide a facility for individual commander training. Outside of training, an automated HFT could be used for mission rehearsal, studies and as an operational 'what if' tool.

7. Acknowledgments

The research described in this paper is funded by the UK Ministry of Defence, under Applied Research Programme 25b.

Acknowledgement must be given to the development team which, in addition to the authors, consists of Richard Gaskin and Bill Jackson from DRA, and Tom Stibbe, Peter Robinson and Alastair Gibb from Logica UK. Without their expertise, this demonstrator could not have been so successful.

Acknowledgement must also be given to the members of the GEKNOFLEXE research programme for their support during this integration.

Finally, thanks must go to Major Adrian Orr and Col John Heard (Retd) for their time and effort fighting the automated OPFOR.

8. References

JPL (1991) "CBS System Design Document", *JPL Publication D-7850*, Jet Propulsion Laboratory, California Institute of Technology.

Lankester, H.G. (1995) "Multi-Application Command Agents", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioural Representation*, Institute for Simulation and Training, University of Central Florida.

Lankester, H.G. & Robinson, P.K. (1994) "GEKNOFLEXE: A Generic, Flexible Model of C^3I", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioural Representation*, Institute for Simulation and Training, University of Central Florida.

Cox, A., Gibb, A. & Page, I. (1994) "Army Training and CGFs - A UK Perspective", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioural Representation*, Institute for Simulation and Training, University of Central Florida.

9. Authors' Biographies

Gary Kendall is a consultant for Logica UK Ltd. He has studied for a BSc and PhD in Physics at King's College London, specialising in learning theory in artificial neural networks. He works in the areas of Artificial Intelligence and Object-Oriented analysis and design.

Ian Page is a Higher Scientific Officer in the UK Defence Research Agency. He has a BSc (Hons) degree in Applied Biology from Liverpool Polytechnic and an MSc in Computer Studies from Sheffield Polytechnic. After spending two years as a member of the GEKNOFLEXE research team, he is now involved with CGFs and synthetic terrain for the Simulation, Training & AI Research group.

Automated Mission Planning in ModSAF

Clark R. Karr, Sumeet Rajput, Jaime E. Cisneros, and Hai-Lin Nee
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
ckarr@ist.ucf.edu

1. Abstract

The ongoing success in realistically representing individual vehicles within Computer Generated Forces (CGF) systems has allowed attention to be shifted to the behavior of groups of vehicles (units) in the battlefield. For the behavior of CGF units to appear realistic, the entities within the units must follow realistic plans. This paper describes a Simulation Based Planning capability developed within the ModSAF CGF system. Simulation Based Planning integrates simulations of candidate plans into the Planning process mirroring the "wargaming" simulation process in human military planning.

2. Introduction

Over the last decade, Computer Generated Force systems have evolved in complexity and detail. The process began with the simulation of simple vehicle dynamics/behaviors and continued through the addition of single entity and small unit (Platoon) behaviors. Now the problem of simulating the Command and Control (C2) process is attracting attention and effort. One facet of the C2 process is mission planning; that is, building a coherent set of actions for subunits and individual entities to accomplish a military goal. The general task of planning has been a topic of research in Artificial Intelligence for many years. Applying the generalized AI Planning techniques within the CGF domain has had some successes and has revealed some problems. Among these problems are the large computational resources required, the difficulty in completely describing the knowledge required, and the lengthy time required for planning.

Lee and Fishwick (1994) propose a Simulation Based Planner (SBP) wherein planning occurs through a two stage process of plan generation and plan evaluation. The evaluation of candidate plans is done through simulation of the plans rather than traditional AI reasoning approaches (e.g. the Multiagent Adversarial Planner (Elsaesser 1991)). The Mission Planner (MP) described here adds a Simulation Based Planning capability to ModSAF. The MP features a Course of Action (COA) Generator that creates multiple COAs (i.e. Plans) and a COA Simulator that simulates each candidate plan. Both the COA Generator and Simulator interact with a Terrain Analyzer for terrain information. "Good" plans emerge as successful simulations while "poor" plans are unsuccessful. The "best" plan is converted to an execution matrix within ModSAF for execution of the plan.

3. Mission Planner architecture

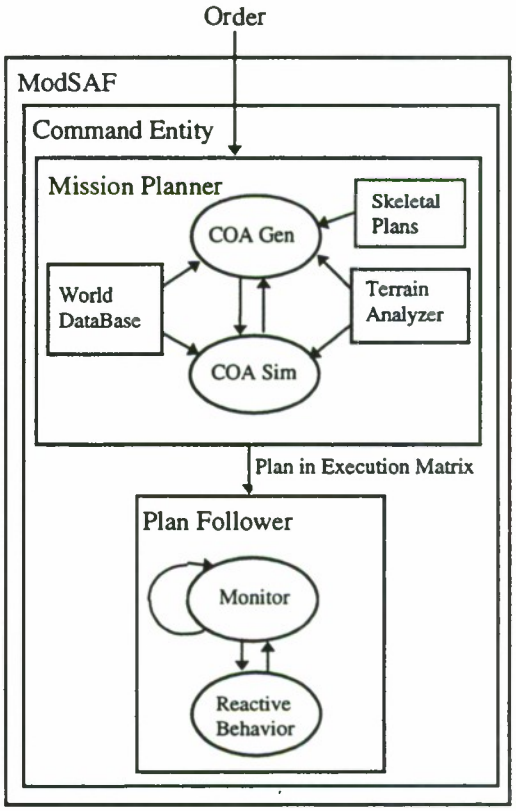


Figure 1: Mission Planner Architecture

Figure 1 shows the architecture of the Mission Planner (MP). The MP is embedded inside a ModSAF Command entity. The input to the MP is given through pull-down menus and is a simple order consisting of:

- the mission, e.g., ASSAULT an objective

- the objective and
- enemy data sets; each set details:
  - an enemy's location
  - the enemy's type (e.g., ARMOR) and
  - the enemy's echelon level (e.g., COMPANY)

3.1 The World DataBase

The World DataBase (WDB) contains information about the battlefield known or believed to be true by the planning entity. This is not a complete spatial representation but a simplified database. It contains:

- location, type, and echelon of enemy units taken from the Order and
- terrain information such as tactical positions and routes, developed during the planning process.

3.2 The Course Of Action Generator

The COA Generator is the module that generates candidate plans from Skeletal Plans (SP) and terrain information from the WDB. The output of the COA Generator is a series of candidate plans which are sent to the COA Simulator for evaluation. The COA Generator is discussed in more detail in Section 6.

3.3 The Course Of Action Simulator

The COA Simulator simulates each candidate plan and outputs a score indicating the relative effectiveness of the candidate plan in completing the mission. The COA Simulator relies on the WDB for the locations of enemy and friendly units. The COA Simulator is discussed in more detail in Section 7.

3.4 The Terrain Analyzer

The Terrain Analyzer (TA) supplies tactical positions (e.g., Assault and Support-By-Fire Positions), routes, and area visibility information. The Terrain Analyzer is discussed in more detail in Section 4.

4. The Terrain Analyzer

The Terrain Analyzer (TA) is a collection of services that provide the Mission Planner with terrain information. The available services are: Area Line of Sight, Route Planning, and Tactical Positions.

4.1 Area Line of Sight (ALOS)

The ALOS module calculates a percentage of visibility between two circular areas. It is given two ordered triples consisting of a point, a radius, and a number of sample points; for example, (p_1, r_1, n_1) and (p_2, r_2, n_2).

Then, n_1 locations are randomly selected within r_1 distance of p_1. Similarly, n_2 locations are randomly selected within r_2 distance of p_2. Finally, $n_1 * n_2$ point-to-point line-of-sight (LOS) checks are done and the percentage of unblocked LOSs is returned. Each LOS check is done three meters above ground level.

4.2 Route Planning

The Route planning component is built on ModSAF's route planning facilities. Given a start location and destination a single route is returned.

4.3 Tactical Positions

Three types of tactical positions are supported: Assault, Support-By-Fire, and Defense. Figure 2 shows two Assault Positions and one Support-By-Fire Position.

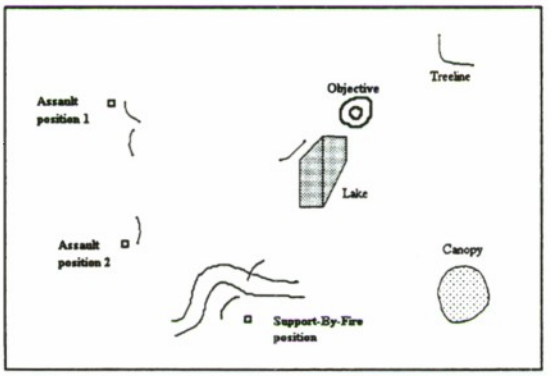


Figure 2: Tactical Positions Example

In Figure 2 the objective lies on a hill. The Support-By-Fire Position is also on a hill and has unblocked LOS to the objective. The Assault Positions are behind treelines and are concealed from the objective.

A grid based approach is used to analyze the terrain. Using the objective as the center, a square $4000m^2$ is determined and divided into square grid cells. The unit boundaries are determined by computing a bounding box encompassing the unit, the objective, and suspected enemy locations. Certain grid cells are

marked as unavailable: cells outside the unit boundaries, cells within grid cell distance of the objective, cells occupied by enemy units, and unreachable cells (e.g., canopies, unfordable water, etc.)

The equation for evaluating grid cells for Assault Positions is:

$$Wap = 0.6 \bullet rt_ratio + 0.2 \bullet slope - 0.5 \bullet (los_obj + los_en)$$

where

- W_{ap} is the Assault Position weight of a grid cell
- rt_ratio is the ratio of the length of a straight line to the route length from the grid cell to the objective
- slope is the slope of the terrain from the grid cell to the objective and
- los_obj is the ALOS to the objective
- los_en is the cumulative ALOS to the enemies

This equation evaluates most positively the cells that have straight line routes to the objective, are "uphill" from the objective, and are concealed from the objective and enemies.

The equation for evaluating grid cells for Support-By-Fire Positions is:

$$Wsbfp = 0.6 \bullet los_obj - 0.2 \bullet los_en + 0.2 \bullet slope$$

where

- W_{sbfp} is the Support-By-Fire Position weight of a grid cell and
- los_obj, los_en, slope are defined earlier

This equation evaluates most positively the cells that have unblocked LOS to the objective, are concealed from enemies, and are "uphill" from the objective.

The equation for evaluating grid cells for Defensive positions is:

$$Wdp = 0.5 \bullet los_obj + 0.6 \bullet los_nearby + 0.2 \bullet slope + 0.2 \bullet obstacle$$

where

- W_{dp} is the defensive position weight of a grid cell
- los_nearby is the average ALOS to nearby grid cells
- obstacle is the ratio of the number of nearby grid cells with obstacles to the total number of nearby grid cells and
- los_obj and slope are defined earlier

This equation evaluates most positively the cells that have unblocked LOS to the objective and nearby terrain, are "uphill" from the objective, and are difficult to reach (have obstacles nearby).

To avoid clustering of resulting tactical positions, when a grid cell has the highest weight among its eight neighbors, the eight neighbors are marked as non-candidates.

The requested number of tactical positions are picked from the highest weighted grid cells in descending order. In the case of equal weights, they are randomly selected.

5. Mission Planner implementation

The MP is implemented as a finite-state machine (FSM) inside a ModSAF task. The user assigns the task to the planning unit from the ModSAF SAFStation. Currently, Assault plans are generated for Company sized units comprised of three or four platoons.

A simplified state diagram of the FSM follows:

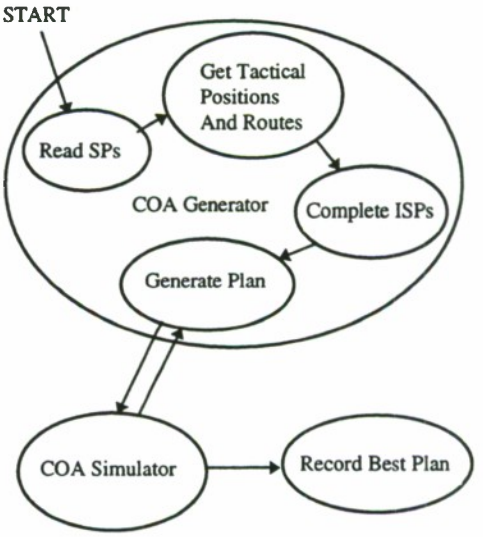


Figure 3: Mission Planning FSM

In state "Read SPs", the Skeletal Plans (SP) for the mission are read from a file and "empty" Internal Skeletal Plans (ISP) are created.

In state "Get Tactical Positions And Routes", tactical positions as specified in the SP and routes are obtained from the TA and stored for later use.

In state "Complete ISPs", the stored tactical positions and routes are added to the empty ISPs to create a list of ISPs. The list of ISPs contain the information to generate the candidate plans.

In state "Generate Plan", a candidate plan is generated. The candidate plan is a "fleshed out" SP with specific locations and routes.

In state "Simulate Plan", a time-stepped aggregate simulation simulates the candidate plan.

In state "Record Best Plan", the best plan for the unit is assigned to the unit's Execution Matrix (Loral 1994a) for execution.

6. Course Of Action Generator

This section describes how candidate plans are generated.

6.1 Reading Skeletal Plans

Skeletal Plans (SPs) are stored in user-defined files. The MP reads from these files the SPs for the task contained in the order. For example, for an Assault task:

```
(ASSAULT
 (ASSAULT_POSITION 4)
 (SUPPORT_BY_FIRE_POSITION 2)
 (SP
    (
     ; subunit 1
     ((MOVE_TO ASSAULT_POSITION)  ; phase 1
     (ASSAULT OBJECTIVE)          ; phase 2
     (CONSOLIDATE_AFTER_ASSAULT ; phase 3
      OBJECTIVE))

     ; subunit 2
     ((MOVE_TO ASSAULT_POSITION)
     (ASSAULT OBJECTIVE)
     (CONSOLIDATE_AFTER_ASSAULT OBJECTIVE))

     ; subunit 3
     ((MOVE_TO SUPPORT_BY_FIRE_POSITION)
     (SUPPORT_BY_FIRE OBJECTIVE)
     (CONSOLIDATE_AFTER_SUPPORT_BY_FIRE
     OBJECTIVE))
    )
    ... Other SPs listed here ...
 )
)
```

Figure 4: Example Skeletal Plan

Figure 4 shows a portion of the ASSAULT SP file and details the first SP. The second and third lines indicate the number of Assault and Support-By-Fire Positions to obtain from the TA. Typically more tactical positions (of a variety) than necessary are requested so that plans using different tactical positions can be evaluated. For example, although a SP may need only one Support-By-Fire (SBF) Position, two or more SBF Positions will be obtained from the TA and individually incorporated into different plans.

Each SP contains a general description of the actions of each of its subunits. The description consists of a list of (task, location variable) pairs, e.g., (MOVE_TO ASSAULT_POSITION). The location variable will be unified with the locations supplied by the Terrain Analyzer during the process of candidate plan generation. Each (task, location variable) pair is treated as a "phase" in the SP.

After the SPs have been read, two data structures are created:

- a mission-specific data structure (for example ASSAULT_DATA for the Assault mission) and
- a list of empty Internal Skeletal Plans.

Initially, the data structures contain basic information such as unit location and location of the objective.

The Internal Skeletal Plans possess the same structure as the Skeletal Plans from which they have been created. They contain lists of phases but these phases do not yet contain any data to create candidate plans.

6.2 Getting tactical positions and routes

The TA is queried for tactical positions and routes.

The number of tactical positions specified in the Skeletal Plan (see Section 6.1) are requested. The generated tactical positions are stored in lists.

Routes are requested between the start location and each tactical position and between each tactical position and the objective. The generated routes are also arranged in lists inside the list containing the tactical positions.

6.3 Completing ISPs

Completing ISPs consists of:

- placing combinatorial information in the phases and
- creating links from the phases in the ISP to the tactical position and route lists.

The combinatorial information determines whether information in each phase changes in the generation of the next candidate plan.

A large amount of memory would be required to put complete terrain data in each phase of the ISP because some information is repeated in multiple phases. To save memory, the data in the phases is linked to tactical positions and route lists inside the mission-specific data structure (Section 6.1). For example, for the SP in Section 6.1 the linkage is shown in Figure 5 .

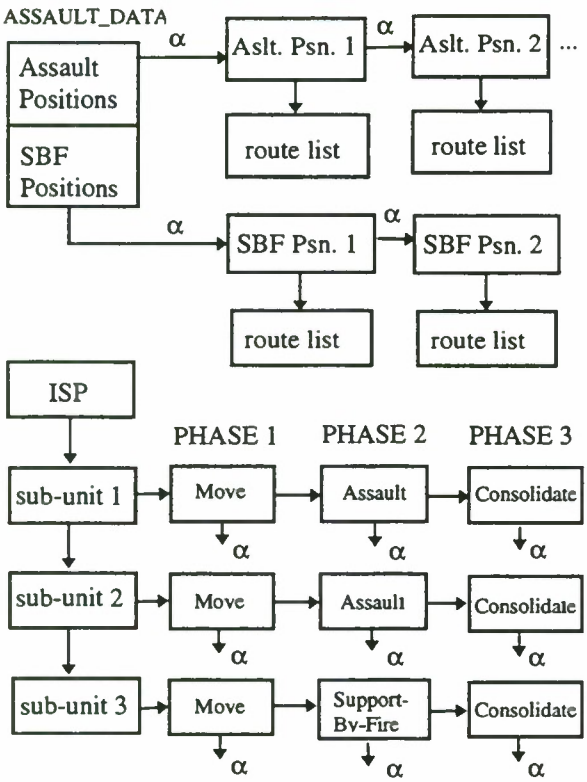


Figure 5: Linkage between ISPs and ASSAULT_DATA

In Figure 5, '∝' is a generic link from a phase to a list (of tactical positions or routes) in the mission-specific data structure. Each tactical position points to the routes available to reach the position.

6.4 Generating candidate plans

At this stage, the MP has a list of ISPs with each task (phase) of the plans linked to lists of tactical positions and routes. For example, a MOVE_TO ASSAULT_POSITION task might be linked to three Assault Positions and the routes to use. Generating all the candidate plans involves generating all the combinations specified in the ISPs. The combinations are generated by advancing the links to tactical positions and routes in an orderly fashion. Each distinct pattern of links to tactical positions and routes represents a new candidate plan.

6.4.1 Adjusting data in subsequent phases

When a phase link pointing to a list of tactical positions is updated the routes to use in subsequent phases becomes invalid and needs to be adjusted.

Suppose, that Phase 1 is a (MOVE_TO ASSAULT_POSITION) phase and the next phase is an (ASSAULT OBJECTIVE). The data link in Phase 1 points to Assault Position *A*. With this setting, the subunit will move from the start location to Assault Position *A* in Phase 1 and then assault the objective from *A* in Phase 2. When the data link in Phase 1 is updated to point to, say, Assault Position *B* Phase 2 must be corrected to Assault along a route from *B*. Thus, whenever the data link pointing to a tactical position is updated, data links to routes in the same and subsequent phases are also updated to make sure that the routes start from the position being moved to in the previous phase(s).

7. Course Of Action Simulator

The COA simulator simulates and evaluates each candidate plan, using a simple, time-stepped, aggregate simulation (constructive). A constructive simulation is used, instead of a continuous time vehicle level simulation, to reduce the time and resources to simulate each candidate plan.

Each time step consists of three stages: Move, Look, and Shoot. That is, all units are moved, all visibility determinations are made, and finally, all combat is executed.

All the phases in a plan are synchronized, and a plan is simulated by executing one phase at a time. When a unit finishes its task ahead of other units, the unit changes to a hasty defense until all the other units have finished their tasks. When all tasks in a phase are complete, the COA Simulator transitions to the next phase.

7.1 Move stage

Vehicle type, maximum vehicle speed and the Operational Activity (OA) of a unit determine how fast a unit moves. The maximum unit speed is taken to be the maximum speed of the slowest vehicle in the unit. The unit speed is adjusted by the operational

activity in which the unit is currently involved. For example, a unit performing a Bounding Overwatch will move slower than the unit performing a Hasty Attack.

Information about enemy units comes in the order and is stored in the WDB rather than being obtained from the simulation ground truth. This mirrors the "real world" where planning must use intelligence information rather than ground truth. The unit speed for enemy units is determined from table 1, based on the unit's echelon type.

Echelon type	Speed (m./min.)
Infantry	90
Mechanized infantry	480
Armor	480
Artillery	480
Supply	480

Table 1: Echelon speeds

Table 2 is used to modify the speed of a unit:

Operational Activity	Speed Adjuster
Move	0.5
Assault	0.8
Attack By Fire	0.8
Hasty Defense	0.0
Road March	0.6
Travel Overwatch	0.5
Bounding Overwatch	0.2
Overwatch	0.2
Hasty Attack	0.8
Deliberate Attack	0.8
Support-By-Fire	0.0

Table 2: Modifying a unit's speed

Once the unit speed is calculated, the simulation time step is used to determine how far the unit will advance along its route. If in a time step the unit can go past one of its route points, the unit will "corner" the route point, and end up on a new leg of its route.

7.2 Look stage

The *look* stage determines visibility between units at their new locations. Due to the aggregate nature of the simulation, only the center of mass of each unit is maintained. Thus, Line-Of-Sight (LOS) between units' centers of mass could be unrealistic. Instead, a percentage of LOS from circles centered at the centers of mass is determined. Table 3 shows the radii by echelon level:

Echelon level	Radius in meters
Battalion	1000
Company	300
Platoon	125
Vehicle	20

Table 3: Area LOS circle radii for echelons

The percentage of LOS values are used in computing how much damage units take.

7.3 Shoot stage

In contrast to using a realistic damage model, we have chosen a simple combat model. In contrast to Lanchester equations (Taylor 1983), this combat model does not reduce the strength of the combatant units. Rather, "damage" accumulates throughout the simulation run and the total damage is used to evaluate the plan. This approach eliminates some of the artifacts introduced by using a time-stepped, aggregate simulation at small unit sizes.

The damage is calculated with these equations:

$$D_i = \frac{S_s * E_s * FP_s * \%LOS_{s \to t}}{E_t * FP_t * Enemy_units_seen_s}$$

where

- D_i is the damage this time step.
- S_s is the strength (number of vehicles) of the shooting unit.
- E_s is the effectiveness of the shooting unit based on its OA (Table 4).
- FP_t is the target unit's firepower (Table 5).
- $\%LOS_{s->t}$ is the percentage of area line-of- sight from the shooting unit to the target unit
- E_t is the effectiveness of the target unit based on its OA.
- FP_s is the shooting unit's firepower.
- Enemy_units_seen is the number of enemy units seen by the shooting unit.

and

$$D_t = \sum_{i=1}^{\text{length of simulation}} D_i$$

where

- D_t is the accumulated "damage" for the target unit.

These equations show the factors that are incorporated into the damage calculation. The "strength" of the shooting unit is adjusted by five factors:

1. The visibility to the target ($\%LOS_{s->t}$).
2. The effect of the unit's current OA (E_s).
3. The firepower of the unit (Fp_s).
4. The effectiveness and firepower of the target.
5. The firepower spread among visible targets.

A unit's effectiveness is determined by its OA (U.S. Army, 1986):

Operational Activity	Effectiveness
Move	0.2
Assault	1.0
Attack By Fire	1.0
Hasty Defense	1.0
Road March	0.1
Travel Overwatch	0.4
Bounding Overwatch	0.5
Overwatch	0.6
Hasty Attack	1.0
Deliberate Attack	1.0
Support-By-Fire	1.0

Table 4: Operational Activity effectiveness

A unit's firepower is determined by its echelon type.

Echelon type	Fire power
Infantry	0.2
Mechanized Infantry	0.7
Armor	1.0
Artillery	0.9
Supply	0.05

Table 5: Fire power of echelons

As mentioned before, information about enemy units comes from the order, so unit strength is determined by the unit's echelon level:

Echelon level	Strength
Battalion	40
Company	12
Platoon	4
Vehicle	1

Table 6: Enemy unit strength

The final score of the plan is the difference between the damage done to enemy units and the damage taken from enemy units:

$$E_p = \sum D_t - \sum D_u$$

where

E_p is the score for the plan.
$\sum D_t$ is the damage to all enemy units and.
$\sum D_u$ is the damage received by unit u.

8. The Plan Follower

The best plan is given to the Plan Follower (PF) for execution. The PF is ModSAF's Execution Matrix controlled mission constructing facility (Loral 1994b).

The best plan appears in the unit's Execution Matrix and is "on order" for execution. Upon receipt of the "on order" signal from the user the first phases for all the subunits are executed. Phases are synchronized so that subunits do not start executing a subsequent phase until all the subunits complete the tasks in their current phase.

ModSAF causes the subunits to take reactive behavior if they are threatened such as by an enemy presence. The reactive behavior may consist of "scrambling" for cover when an enemy is sighted. Options allow the user to override the reaction causing the subunits to resume execution of the plan.

9. Experience with ModSAF

ModSAF 1.3 was used as the platform for development. The CTDB (Compact Terrain Data Base) was used for terrain reasoning purposes. ModSAF library functions were used whenever possible. The capability of generating candidate plans with alternative routes between locations was not exercised because the ModSAF route planner returns only one route between positions.

Display time during COA simulation can be significant and may slow down execution causing non-real time execution of the system.

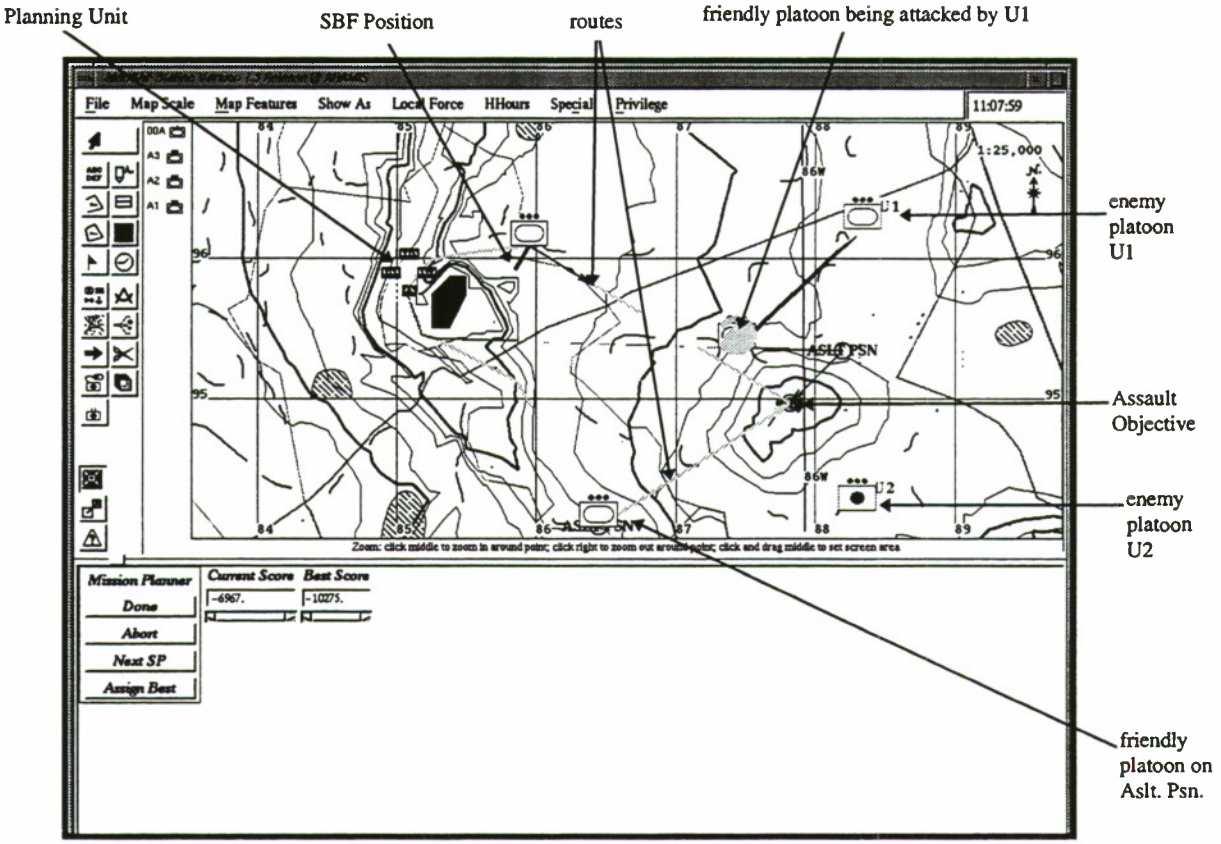


Figure 6: Candidate Plan Simulation in ModSAF

We experienced this problem when we attempted to display all the tactical positions and routes that were part of a candidate plan in a single state of the MP FSM. Modifications had to be made to the MP FSM to spread the display across states to alleviate the problem.

10. Results

Skeletal Plans (SPs) for the Assault mission have been implemented (see Figure 4). For the SP shown in that figure, the COA Generator and Simulator generate and simulate 32 candidate plans.

Figure 6 is a screen dump that shows a candidate plan being simulated for an Armor Company. The Company is positioned to the left of a lake and is planning an Assault mission on the hill shown on the right side of the picture (Assault Objective). The plan being simulated in Figure 6 involves a Company of three platoons. One platoon provides supporting fire. The other two platoons assault the objective from different Assault Positions. Figure 6 shows one platoon at the Support-By-Fire (SBF) position, another at an Assault Position, and a third being hit by enemy fire (represented by a circle) while on its way to the second Assault Position. The routes being used by the Platoons are visible in the background.

The editor below the tactical map display shows the current score of the plan and the score for the best plan so far. The greater the score, the better the candidate plan. The score for the current plan (-6967) is greater than the score for the best plan so far (-10275). If the final score for this plan is greater than the "best plan so far", it will become the new best plan; otherwise it is discarded.

11. Conclusions and future work

CGF systems are becoming increasingly complex and greater attention is being paid to representing the behavior of units. Mission Planning is an essential component of the Command and Control process. The Simulation Based Planning approach described here mirrors the human decision making process wherein different courses of action are evaluated and the "best" one is chosen. This approach provides an alternative or adjunct planning mechanism to the traditional AI knowledge intensive approaches. The

Mission Planner (MP) has been implemented in ModSAF and tested for a Company Assault mission.

There are several opportunities for future work with this system. The MP consists of three primary modules: the Terrain Analyzer (TA), COA Generator, and COA Simulator. Each component can be improved and extended; some ideas are discussed below.

The TA currently returns a single route between locations. Thus, candidate plans with alternative routes between locations cannot be generated. Karr et. al. (1995) describes a Unit Route Planner that has the ability to generate multiple routes between locations. Implementing this route planning capability in ModSAF (as part of the Terrain Analyzer) would increase the variety of candidate plans.

The TA's grid based analysis of tactical positions can be improved by a more thorough analysis based on computational geometry.

The COA Generator creates candidate plans by systematically generating all the combinations specified in the Skeletal Plans. Applying AI planning techniques during this process to detect and eliminate from consideration impossible plans could significantly decrease the computational expense of simulating "bad" plans.

The COA Simulator could simulate plans more realistically by applying a more sophisticated combat model (e.g., one using Lanchester equations described in (U.S. Army 1986)). Additional criteria can also be incorporated into plan evaluation. For example, considering the time of completion of a plan. In real military planning, the time of completion of a plan is important

This project has shown that realistic plans can be created and evaluated with a Simulation Based Planning approach. It seems reasonable that this approach could be applied within knowledge intensive planning approaches to reduce the complexity and breadth of knowledge required. That is, a COA Generator/Simulator module could identify (through simulation) better plans without encoding specific knowledge to make fine distinctions among plans.

12. References

Lee, J. J. and Fishwick P. A. (1994). "Simulation-Based Planning for Computer Generated Forces", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida, pp 451-460.

Elsaesser, C. and MacMillan, T. R. (1991) "Representation and Algorithms for Multiagent Adversarial Planning", Technical Report MTR-91W000207, MITRE Corp. pp 1-56.

Loral (1994a). "LibExecmat documentation", Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1994.

Taylor (1983). *Lanchester Models of Warfare*, Volume 1, 1983.

U.S. Army (1986). "The division 86 Tank Company SOP", *Coordinating Draft*, U.S. Army Armor School, May 1986.

Loral (1994b). "ModSAF User Manual", Loral Advanced Distribution Simulation, Cambridge, Massachusetts, September 30, 1994.

Karr, C. R. and Rajput S. (1995). "Unit Route Planning", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida.

Winston, P. H. (1992). *Artificial Intelligence*, Third Edition, Addison-Wesley Publishing Company, 1992.

13. Author biographies

Clark R. Karr is a Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Sumeet Rajput is an Associate Engineer in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Jaime E. Cisneros is a Research Associate in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Cisneros has a Masters of Science degree in Computer Science, and is currently working on a Masters of Electrical

Engineering. His research interests are in the areas of Natural Language Understanding, Machine Learning, and Computer Generated Forces.

Hai-Lin Nee is an Assistant Engineer in the Intelligent Simulated Forces Project at the Institute for Simulation and Training. Mr. Nee has a Master of Science degree in Computer Engineering. His research interests are in the areas of Expert Systems, Object Oriented Analysis, and Computer Generated Forces.

Multi-Application Command Agents

Helen Lankester
Software Engineering Centre,
Defence Research Agency, Fort Halstead
Sevenoaks, Kent TN14 7BP, UK.
hlankester@dra.hmg.gb

1. Abstract

The UK Defence Research Agency has set up a collaborative research programme, called CARE, aimed at developing a command decision-making function for use by a number of different applications. This work will build from the command agents developed for the GEKNOFLEXE project, making them accessible to other simulation systems, extending their use and enhancing the tools provided for users and developers.

Proof of principle demonstrators have been produced; these link the existing GEKNOFLEXE simulation to a Higher Formation Trainer and a divisional level war game used for operational analysis. This has allowed command agents to successfully control units within these simulations. A more open and modular approach is required which provides for a better interface between command agents and simulations.

An outline architecture for the common command agent software has been designed. This consists of a command agent shell and an environment in which multiple shells can operate and interact with entities within simulations. The CARE programme will develop more generic knowledge bases for the command agents so that they can operate in a wider range of scenarios and simulation systems.

2. Introduction

A number of research projects within the UK Defence Research Agency (DRA) have recognised the need for an explicit representation of command and control, in order to meet their differing requirements. Four of these programmes have joined together to share the effort and cost of this by developing a common software model of command decision-making. This collaborative programme is called the DRA Command Agent Research (CARE) initiative.

This paper describes the current use of command agents within the projects contributing to CARE and discusses their requirements for the common software model. The issues associated with implementing such a model and an architecture to provide the required facilities are outlined.

2.1 Command Agents

Work under the CARE programme is based on the command agent function developed as part of the GEKNOFLEXE system. This was presented at the Fourth Computer Generated Forces and Behavioral Representation conference (Lankester and Robinson, 1994). Command agents are used to represent the decision-making nodes within the command hierarchy. Each command agent typically represents a command post which is able to make decisions and interact with other command agents and entities within a simulation thus controlling battlefield operations.

Command agents are entities within the battlefield simulation and so are subject to battlefield effects. The decision-making capability of a command agent is represented using a set of rules describing the tactics which that command agent needs to use to perform its role in the battle and an object structure in which it stores its perception of the battlefield.

Each type or class of command agent has its own set of rules and domain knowledge structures which define its behaviour. Instances of command agents are created which have their own perceptions of the battlefield which they maintain as the simulation progresses.

2.2 Terminology

Command Agent - software which represents the command and control decision-making of a command post.

GEKNOFLEXE (GEneric KNOwledge-based FLEXible Enemy) is a project which has produced a tool set for modelling command and control as a set of communicating command agents.

CARE (Command Agents REsearch) is a DRA initiative to develop a generic command agent software function and associated facilities. It is based on GEKNOFLEXE.

3. Applications of Command Agents

Large numbers of human players and controllers are

required to operate many war games and simulations to represent command and control. Providing them is costly and it is becoming increasingly difficult to find suitable military personnel to staff such systems. This has led several different projects to investigate the use of command agents within their simulations.

3.1 Current use of GEKNOFLEXE Command Agents

The GEKNOFLEXE project has been using and developing command agents for three years to characterise command and control so that its effectiveness in different situations can be studied. Two major scenarios have been developed and a number of studies have been undertaken using them.

Two simulations which require a command and control model have also used the GeKnoFlexE system to provide a proof of principle demonstrator to show how they could use command agents. These demonstrators are described below.

3.1.1 Command Agents for a Higher Formation Trainer

One of the demonstration systems links the existing GEKNOFLEXE software to the Corps Battle Simulation (CBS) and is described in more detail in Kendall and Page (1995). This demonstrator has successfully used the CBS generic interface to enable the GeKnoFlexE command agents to control the opposition forces within CBS thus reducing the number of controllers required. Current research in this area is focused on reducing the controller workload in the UK Army's new Higher Formation Trainer, ABACUS.

3.1.2 Command Agents for Operational Analysis

The GEKNOFLEXE software has also been connected to a divisional level war game used for operational analysis. The GEKNOFLEXE system acts as a virtual player on the simulation's network so that command agents are able to control recce units. Recce units were automated because they are used extensively early in the simulation, when the external players controlling the game are not very familiar with its operation. Future work is aimed at reducing player workload and ultimately the number of players by automating parts of other sub-functions, e.g. manoeuvre and engineers, thus helping to solve the problem of finding sufficient military players for the game.

3.2 Common Command Agent Software

The GEKNOFLEXE system has its own battlefield simulation in which the command agents operate. The simplest way to adapt the GEKNOFLEXE system to get its command agent to control units in another simulation is to make the GEKNOFLEXE battlefield simulation emulate the target simulation. In this way the command agents can continue to work within the simulation designed for them but have their orders forwarded to the new one. This approach has been used to produce both of the demonstration systems described above.

The current approach to using GEKNOFLEXE command agents in other simulations is restricting further development and running two battlefield simulations is an unnecessary overhead. A more open and modular approach is required to command agents, one which allows a better interface between command agents and simulations. Approaching the development of command agents in this way will enable central development of a command agent software function which can be used by a number of different applications.

The projects described above are contributing to the CARE programme as they all require a similar model of command decision-making. It is clear from the use of the existing system by other projects that GEKNOFLEXE command agents are applicable to other areas and therefore the cost of developing them could be shared. A collaborative program based around one simulation system with a command decision-making model would be ideal, however most of the prospective users of such a system each have their own simulation system specifically designed for their purposes. The element which these simulations are lacking is a representation of command decision making which GEKNOFLEXE command agents offer.

The aim of the CARE project is therefore to extract the GEKNOFLEXE command agents from their current simulation, make them accessible to the other simulation systems as well as extending their use and enhancing the tools provided for the user and developer.

3.3 Other Applications

The approach taken by CARE does not restrict the use of command agents to the current application areas which are supporting the project. CARE's command agents could be used in other training or operational analysis systems as well as in doctrine development and perhaps in the longer term used within a decision support tool. Any simulation which requires a reasonably complex command decision-making capability could make use of the command agent approach.

4. Requirements

The requirements of the different applications are surprisingly similar given their different ultimate aims. They all require a representation of command decision making and the facilities to interact with the command agents during operation.

4.1 Command and Control Representation

Some of the requirements are already met by the existing GEKNOFLEXE command agents which provide the ability to represent different levels and sub-functions of command. The modularity provided by the GEKNOFLEXE command agent enables the complexity of the C2 model to be represented appropriately.

With increasingly sophisticated command agents the need for processing power quickly increases and the architecture must take this into account. One solution to this problem is to allow distributed and parallel use of command agents. Command agents are particularly suited to parallel operation as this is exactly how real world command posts operate. The performance of command agents is important as they must operate in synchronisation with the simulation and not affect the overall performance of the system.

A framework is required in which different command and control models can be represented. This is essentially what the GeKnoFlexE tool set already provides with its ability to represent different types of command agent.

Some applications require the command agent's decision-making to be deterministic, for example an operational analysis study comparing different weapon types. On other occasions it may be appropriate to vary the decision-making to represent different commanders so that trainees do not become too familiar with the behaviour of command agents.

4.2 User Interfaces

A major part of CARE is devoted to developing the facilities and graphical interfaces to develop and use command agents. There are three main types of user: the command agent developer, the military validator and military users who act as controllers or players. The user interface must allow users to understand the command agent's behaviour. This is important for verification and validation as well as for run time users, who can then decide if and when they need to alter or take over control from a command agent.

The facilities required by validators and run time users are similar since they must both be able to monitor what command agents are doing. A trace and explanation of the command agent's decision-making process goes some way to achieving this and can also be used for After Action Review. The huge quantity of this sort of trace information in a reasonably complex command agent system can quickly become unmanageable so user definable filters are needed.

4.3 Human Command Agent Interaction

A user of the CARE system must be able to override the decisions made by command agents. This may be to make them act in a non-doctrinally correct manner in order to meet a particular training objective or to ensure that a new weapon type is used in an operational analysis study run. Another reason for wanting to override decisions is to cover areas of knowledge where the command agent is deficient.

A step on from overriding command agents is handing control over to a human controller completely i.e. a 'man in the loop'. The human controller may need to work with command agents as his superior or subordinates. This facility could be used for similar purposes to overriding.

The controller may want to hand back control to the command agent. Whilst a command agent is operating it maintains its own perception of the battlefield and what it is doing on which to base its decisions. The command agent will have to maintain a perception of what is happening whilst the human is in control if it is to be able to take control back.

A controller could also control a command agent's behaviour by acting as its superior and giving it appropriate orders. This introduces the issue of voice communication with command agents.

4.4 Interaction with Simulations

The command agents need to be able to interact with a number of different simulations. Command agents have to give orders to and receive reports from other entities within the simulation. They will also need to be able to move the command vehicles which are their physical representation in the simulation and gain access to information from the simulation, e.g. time and the terrain database.

4.4.1 Magic Moves

Simulations requiring a C2 model often have a 'magic move' facility which is used to correct human errors, for example, players often forget about bridging units until they need to use them. This sort of mistake should not hold up the whole game so corrections are made by 'magically' transporting the relevant units into position. Magic moves are also used to meet specific objectives but in a more dramatic way than

overriding decisions. A magic move may require the perceptions of some of the command agents to be changed, just as a human controller's perception would need to be changed.

Changing the perception of command agents when a bridging unit has been magic moved will be fairly easy and have few side effects. If a whole brigade had been magic moved into a new location the effect on the command agents would be more severe. Their perceptions need to be changed so that they have a realistic picture of the battlefield and do not undermine the intention behind the magic move.

4.4.2 Re-execution, Rewind and Fast Forwarding

Other simulation functions which the command agents will need to handle include re-execution, rewinding and fast forwarding. Re-execution and rewinding require sufficient data to be stored for the command agents to recall their perceptions from a point earlier in the simulation and be able to start again from this point. This kind of data is also required for After Action Review. Fast-forwarding the command agents' actions has further performance implications, making the ability to use command agents in parallel more important.

The scenarios which are used in simulations often start when the forces have, or are about to, come into contact. A controller of the simulation will have to give each of the players an idea of their situation. The same is true for command agents. The fast forward facility could be used to allow the simulation to be started slightly before the forces have come into contact so that the command agents build up their own detailed perception of the battlefield. Being able to fast forward the command agents will also aid development and validation.

4.4.3 Use of Simulation Models

Command agents currently use models within the GEKNOFLEXE simulation, for example to send messages or plan routes. The command agents will continue to need to exploit models within the target simulation directly, for example the communications and terrain models. The effects of various simulation models, such as attrition will also have to be considered. The limitations of the simulation will need to be handled by the system, for example, the simulation's congestion model may prevent command agent from deploying their units as they would wish.

4.5 Generic Knowledge Bases

So far command agent knowledge base development has been centred around specific scenarios to constrain the required functionality. The knowledge bases required by command agents are very costly to produce, as considerable effort is required to write and validate them, and they are currently limited to specific types of scenario. The CARE programme has been tasked with designing knowledge bases which are as reusable as possible. Command agents need knowledge bases which allow their operation in a range of scenarios as well as with a number of different simulations.

4.6 Standards

Appropriate standards need to be adopted so that the CARE system can be used as widely as possible and interact with other computer generated forces, e.g. ModSAF. In this context adoption of the ARPA's (Advanced Research Projects Agency, US) emerging Command and Control Simulation Interface Language (CCSIL) will be considered.

5. Command Agent Architecture

An architecture and approach to developing the common command agent software has been designed in outline. It consists of a command agent shell and an environment in which multiple shells can operate. The command agent software will interact with each simulation via a mediating process which provides an interface between the simulation and the command agents.

5.1 Command Agent Shell

The command agent shell is a command agent without a knowledge base. It provides the facilities for developing and using a command agent. This part of the CARE work will reuse the inference engine, knowledge representation and knowledge base tool used in the GEKNOFLEXE project.

5.1.1 Support Functions

The GEKNOFLEXE command agent knowledge bases currently use a number of support functions to aid decision-making. Some of these functions access the simulation directly to effect actions such as establish the current time and use the terrain database. Some support functions may even access ground truth directly to simplify a process, for example data fusion. Whilst this is clearly allowing the command agent to have access to non-real world information, it significantly reduces the complexity of the command agent, allowing a wider representation of the battlefield for a given amount of resources. It is also thought that command agents will continue to need more than real world information to model command decision-making effectively.

5.1.2 Message Interpreter

As well as facilities for building and executing

knowledge bases the command agent shell will need to include a message interpreter which takes in messages from other command agents and entities and sends out messages from the command agent. The message interpreter will forward messages to the appropriate part of the command agent and queue messages if it is busy.

5.1.3 User Interfaces

The developer's interface is largely present in the form of the knowledge base tool already in use by the GeKnoFlexE project. This tool allows the user to develop rules and objects in a menu driven system. The same tool provides facilities for tracing through rule executions and domain knowledge structures.

The knowledge base tool is a good starting point for the development of facilities to allow users to understand the behaviour of command agents. The interface needs to be able to convey to the user the information he wants as quickly as possible so that he is able to follow what the command agent is doing. Figure 1 shows what this interface might look like.

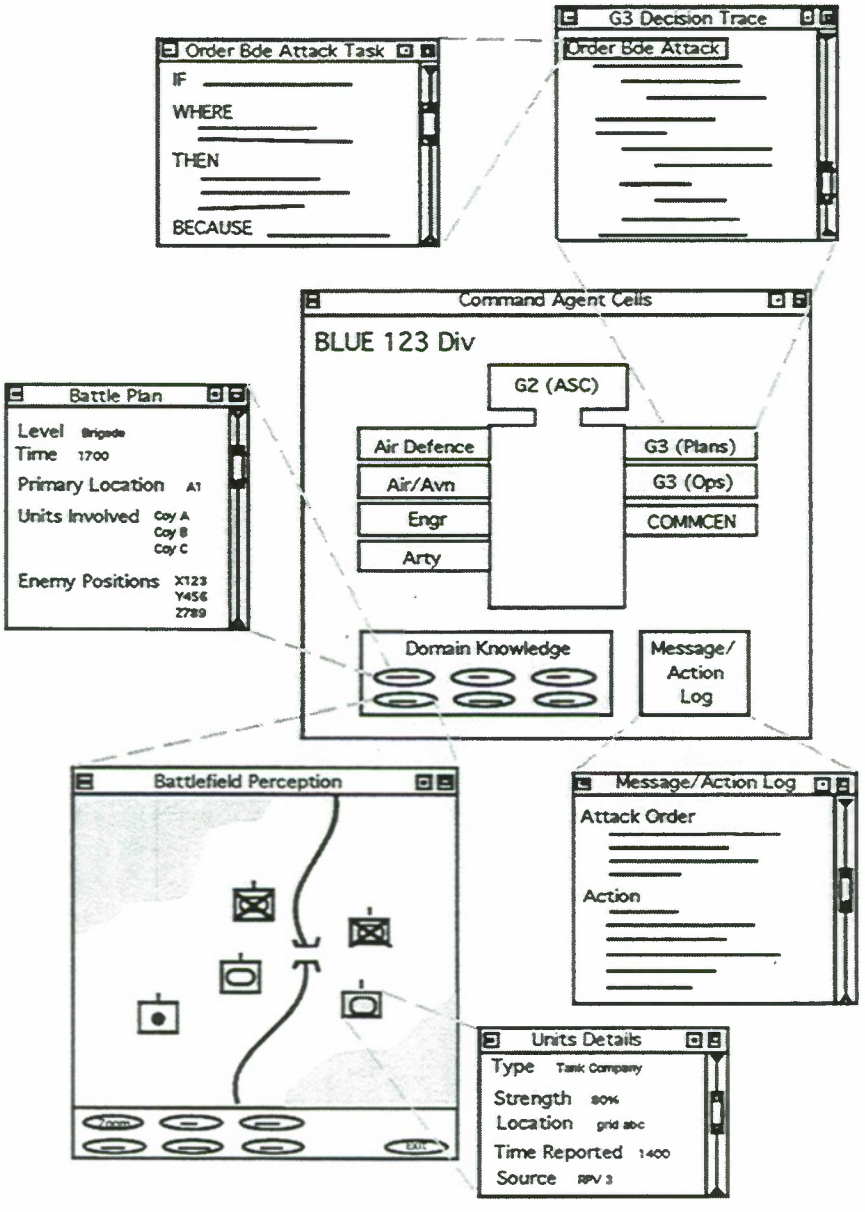


Figure 1: Window Based User Interface

A multi-window system allows different types of information to be displayed simultaneously, for example a map display of the perception of a command agent and a trace of the decisions being made using that perception.

It is important that information is displayed to the user in a consistent and familiar format. The command agent cells window in figure 1 represents the command agent in a similar way to that used to represent command posts in officer's handbooks.

The quantity of information available to the user is so large that the interface must provide facilities to enable him to manage it easily. This can be achieved using user definable filters, which only show the user the information he has requested, and also by using the multi-window approach to provide top level information, which can be selected and focused in on as necessary. An example of this might be for the user to monitor a high level action and message log. When he sees something happening of particular interest, he could then select it and look at the decision-making trace and perhaps even the rules themselves. This approach could also be used for displaying the domain knowledge; a map display showing the command agent's perception of the battlefield could be displayed and units on the map selected for more detailed information.

5.2 Command Agent Environment

The command agent environment will contain a configuration facility which sets up the command agents to be used in a particular simulation run and defines their relationship to each other and the processes in which they are located. This configuration information will need to be maintained persistently within a database.

The command agent environment controller will be responsible for passing messages between command agents and routing messages to and from the simulation. Messages from the simulation may be from entities modelled by the simulation or the results of support function queries. Orders and reports will have to be sent using the simulation's communications model (if it has one). This will allow messages to be delayed or lost as the communication model dictates.

The environment controller will have to ensure that the command agents' decision-making is keeping up with the simulation. This is particularly necessary when command agents are making decisions over a period of simulation time. The controller also needs to be able to take account of the effects of attrition on command agents.

5.3 Simulation Interface

The interface between the system developed by the CARE initiative and the simulation is referred to as the mediating process. This process is made up of two parts; a generic part common to all simulations and a simulation specific part. It is hoped that this will ultimately be compliant with emerging standards like CCSIL. However, CCSIL provides only real world messages which may not be sufficient for current command agents.

This mediating process is responsible for translating message formats from the simulation to the command agents and vice versa. The mediating process needs to monitor the simulation so that it can provide command agents with realistic battlefield reports where these are not provided by the simulation. It also extracts information from the simulation required by the support functions and by the command agent environment controller. This will include simulation time and details of magic moves and rewinds.

In order for the GEKNOFLEXE system to use the new command agent facilities a simulation specific part of the mediating process must be produced. This is seen as relatively simple, since the simulation has been designed to interact with command agents. Building an interface to other simulations is likely to be more complex. Most training and operational analysis simulations have been designed to work with people so a command agent should be able to interface in a similar way to the human. However, there are problems associated with their different abilities to reason with information, for example a human controller can interpret a raster map where as a command agent would have difficulty!

5.4 Physical Layout

Provision has been made to encompass distributed and parallel processing techniques in order to achieve the required levels of command agent performance. Figure 2 shows the physical layout of the command agent architecture. There will be a command agent environment controller and command agent shell on each process running command agents. Each command agent shell will run a number of command agent instances.

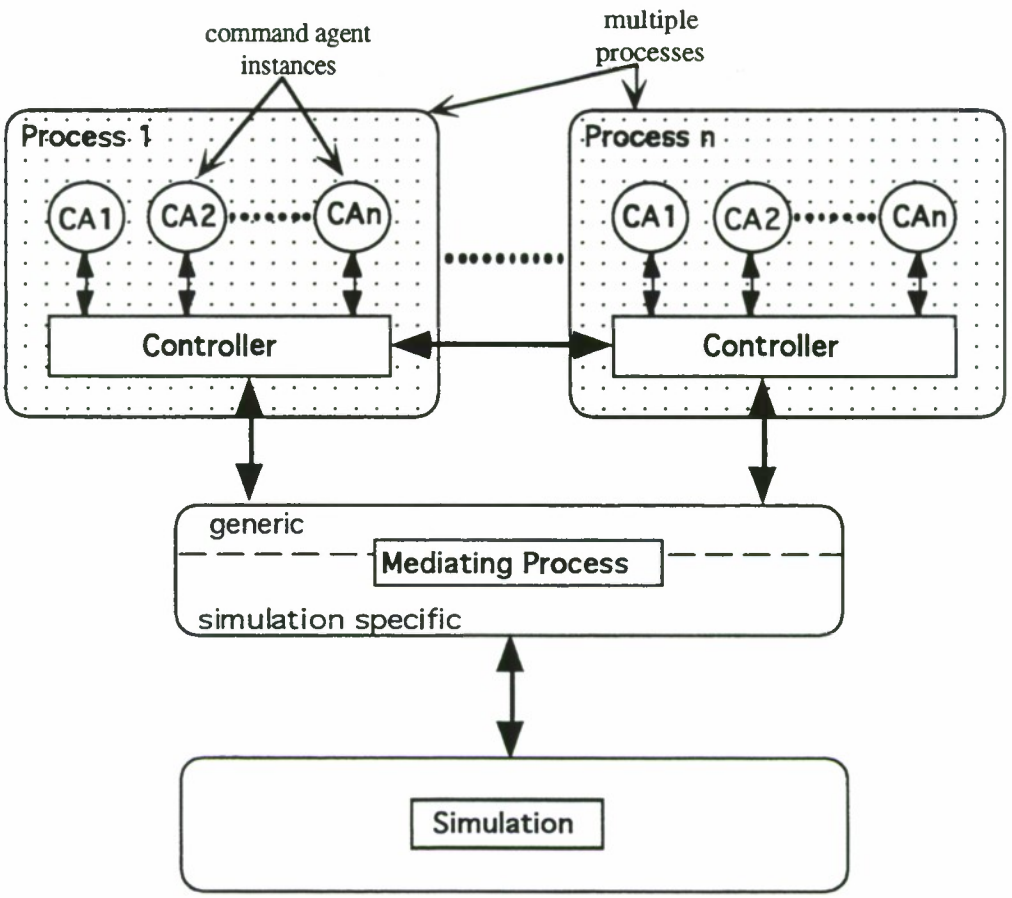


Figure 2: The CARE System Architecture

6. Knowledge Bases

The command agent shell outlined in section 5.1 describes the software which is used to develop and run command agents. Each type of command agent requires the development of a knowledge base to provide it with the appropriate decision-making capability.

The knowledge bases developed for the scenarios used in the GEKNOFLEXE project will be used initially whilst the CARE architecture is developed. These knowledge bases have been developed for specific concept of operations and force structures. This means that the command agents can only be applied in a limited range of scenarios. As figure 3 shows this representation is better than the more conventional scripted models of command and control but not nearly as flexible as a human would be. If the command agents are to be reused to a greater extent they will need to be more flexible so they can cope with a wider range of scenarios.

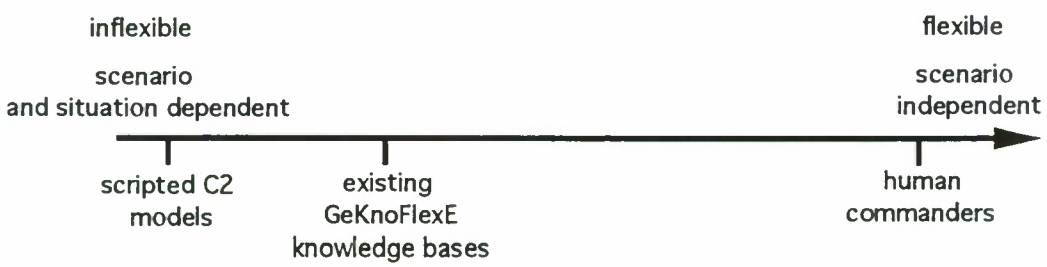


Figure 3: C2 Modelling Flexibility Scale

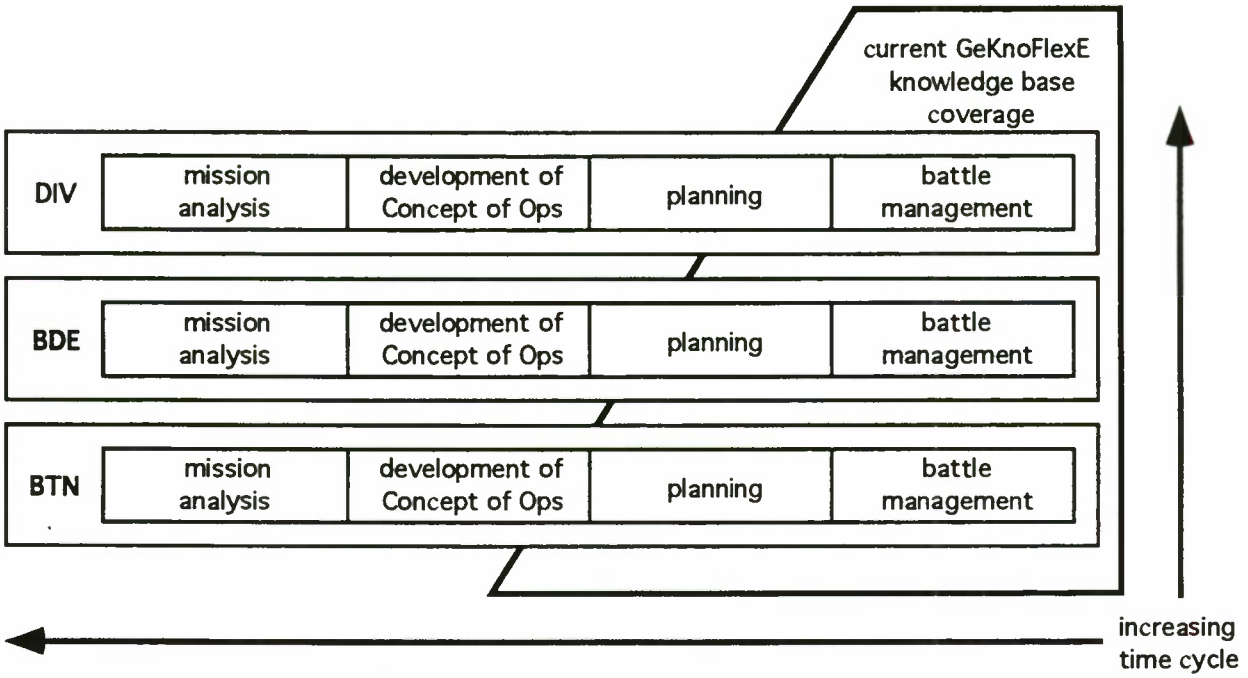


Figure 4: Military Decision-Making Currently Represented Using GeKnoFlexE

The CARE knowledge base development work will build on the experience of the GEKNOFLEXE project to expand the applicability of the command agents. Figure 4 shows the types and levels of military decision-making currently represented in GEKNOFLEXE. The more reactive types of decision-making have been covered in more detail. Command posts at the lower levels have to make decisions over smaller time periods than the higher ones. This tends to be easier to represent and needs to come into play earlier on in a simulation run.

The less reactive types of decision-making need to be represented. This includes the kind of knowledge required to develop the concept of operations. This will enable the higher levels of command,, for example at divisional level, to be represented in more detail and therefore provide longer running, more varied scenarios than have currently been achieved.

As well as operating in a wider range of scenarios, command agents must cope with differences in the behaviour of distinct simulations. Only then will it be possible for them to be used more generally by different applications.

7. Work Programme

A feasibility and scoping study for the collaborative command agents research work was completed in October 1994 (Heard et. al. 1994). This study showed that the establishment of a collaborative project to develop a generic command agent software function was both feasible and likely to provide significant savings in comparison to each project pursuing independent programmes. As a result of the study a project has been set up and initial design and development work started.

A spiral development approach has been taken to the development of the command agent software. The first spiral consists of developing a prototype system which will be used to further identify the requirements of the applications and research possible solutions. Early work will remove the command agents from the GEKNOFLEXE system and produce a mediating process to link them back to the simulation. It is hoped that the CARE prototype system will be demonstrated linked to ABACUS by the end of 1995. This first phase will not develop any further knowledge bases but will use the existing ones developed for the GEKNOFLEXE project.

The second spiral will develop the full command agent system including a comprehensive set of tools and interfaces for users. It will also provide a wider and more applicable set of knowledge bases which each of the programmes can use.

8. Conclusions

GEKNOFLEXE command agents have successfully controlled units within three different simulations, reducing controller/player workload. These simulations are used for different applications ranging from training to operational analysis, and yet they all require a similar command decision-making model and associated facilities for users. The command agent capability provided by GEKNOFLEXE has been shown to be a suitable basis from which a common command agent software function and facilities can be developed for use by different applications.

The CARE programme was set up to develop the common command agent software, making the use of GEKNOFLEXE command agents more accessible. The command agents will be extracted from their current simulation and improved facilities for users and developers provided. Command agents will also be made more reusable by extending their knowledge bases to cover a wider range of scenarios, battlefield functions and use within different simulation systems.

9. Acknowledgements

The CARE project is funded by the UK Ministry of Defence. The project is conducted by the Software Engineering Centre of the UK Defence Research Agency with 'Rainbow Team' support from Logica UK.

10. References

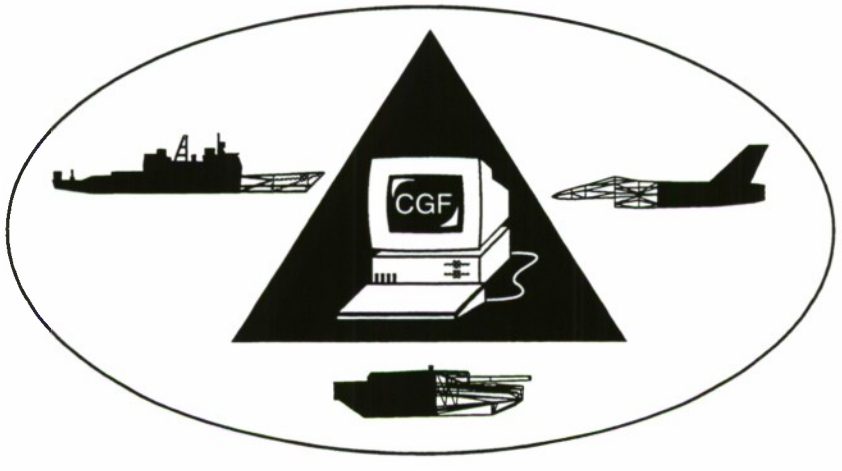
Heard, R. J. B., Lankester H. G., and Robinson, P. K. (1994) "Command Agents research feasibility/scoping study report" *Technical report DRA/CIS/CSS1/TR94006/1.0*, Defence Research Agency.

Kendall, G., and Page, I. (1995) "An Automated CBS OPFOR", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation.*

Lankester, H. G., Robinson, P. K. (1994) "GeKnoFlexE: A Generic, Flexible Model of C3I", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.*

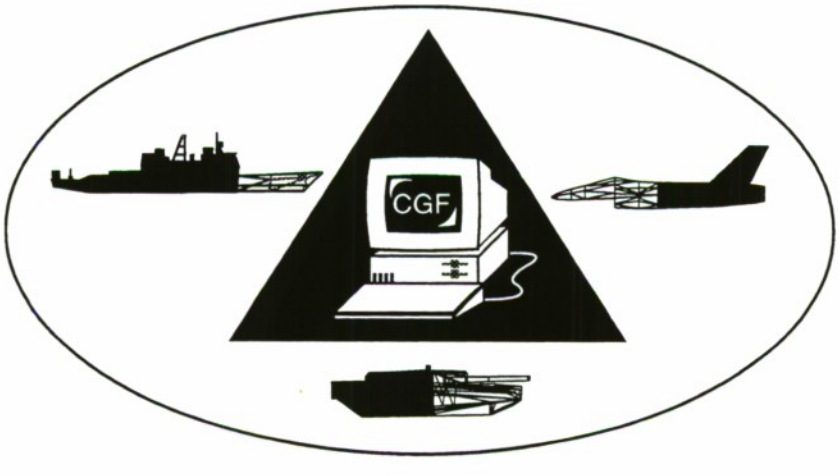
11. Author's Biography

Helen Lankester is a Higher Scientific Officer in the UK Defence Research Agency - an agency of the UK Ministry of Defence. She graduated from Sussex University with a BSc degree in Computer Science in 1991. Since then she has worked on the GEKNOFLEXE project and a rule-based movement model. She is now the Technical Leader of the CARE project.


CGF

Session 4b: VV&A

Meliza, ARI
Perneski, Loral ADS
Thomas, AMSAA


CGF

Measuring Entity and Group Behaviors of Semi-Automated Forces

Larry L. Meliza and Eric A. Vaden
U.S. Army Research Institute
Simulator Systems Research Unit
12350 Research Parkway, Orlando, FL 32836-3276

1. Abstract

The Unit Performance Assessment System (UPAS) was developed to measure the behavior of manned units in distributed interactive simulation (DIS) exercises, but measures of performance used to assess manned units can also be applied in measuring the behavior of computer generated forces (CGF). This paper describes the use of the UPAS as part of an effort to compare the behavior of Semi-Automated Force (SAFOR) Version 4.3.3 with Modular SAFOR (ModSAF) Version 1.2. UPAS data displays were used to assess how well the actions of individual entities were controlled by mission, enemy, time, and terrain variables. These displays were also used to assess how well entities worked together as part of armor or mechanized infantry platoon-level organizations in performing collective tasks described in the Mission Training Plan (MTP) document appropriate to each type of unit. Behavioral assessments in the individual entity mode included examinations of the effects of terrain on vehicle speed, rate of fires when confronted with target arrays, land navigation, and scanning behavior. Group behaviors assessed at platoon level included quality of formations, smoothness of transitions among formations, observation (scanning) within assigned sectors, reaction to contact, and the conduct of an assault. The two CGF systems differed from one another in terms of various measures of performance. Overall, both types of CGF displayed inadequate sensitivity to the mission, enemy, time, terrain, and troop (METT-T) variables that should be controlling CGF behavior.

2. Introduction

The Unit Performance Assessment System (UPAS) was developed to measure the behavior of manned units in distributed interactive simulation (DIS) exercises (Meliza, Bessemer, and Tan, 1994), but measures of performance used to assess manned units can also be applied in measuring the behavior of CGF. The UPAS has previously been used to compare the behavior of manned units with that of one version of CGF (Vaden, Meliza, and Johnson, 1994; Mengel, 1994) in an effort known as Summer Exercise I (SUMMEX I). This paper describes the use of the UPAS as part of an effort to compare the behavior of two CGF systems Semi-Automated Force (SAFOR) Version 4.3.3 and Modular SAFOR (ModSAF) Version 1.2.

A week of exercises were conducted in July of 1994 to compare the behaviors of ModSAF Version 1.2 and SAFOR Version 4.3.3 to determine which system was most appropriate for Synthetic Theater of War-Europe (STOW-E). Many of the tactical scenarios used in comparing the behavior of the CGF were taken from Summer Exercise I (SUMMEX I) in which the behavior of ModSAF Version C was compared with that of manned platoons (Mengel, 1994). These platoon and company level scenarios were supplemented by mini-scenarios in which the focus of observation was a single entity.

Many of the behavioral performance criteria could be applied by the members of the assessment group viewing the action through an out-the-window view from a stealth station (Loral GT-101). In other cases, UPAS data displays were used either as the initial assessment tool or to quantify a differences in performance we believed we were seeing as we observed the out-the-window view. In certain cases the UPAS displays were used to document critical behaviors in a manner that could be readily incorporated into reports. Because ModSAF operated under DIS 2.03 protocols rather than SIMNET protocols during portions of the test, the

UPAS collected network data from the SIMNET side of a protocol translator.

The UPAS data displays included "Snapshots" showing overhead views of the exercise, traces of entity or unit movement over time, data summary tables, and graphs. Graphs and tables were produced using data loaded into two relational database tables, the Pairing Event Table (PET) and the Ground Player Location Table (GPLT). The PET is based on information taken from the SIMNET Fire, Impact, and Status Change Protocol Data Units (PDUs). For each firing event, the PET shows the time, location of firing entity, location of the round or bomb impact, ID of the firer, ID of the target, ID of the type of weapon system, ID of the type of ammunition, the number of rounds fired, the result of the firing event, the range from firer to impact, and the firing event number. The GPLT contains information taken from the Vehicle Appearance and Vehicle Status PDUs. This table identifies the time of the status update, the ID of the entity, the location of the entity in terms of X, Y, and Z coordinates, entity speed, direction of movement, orientation of gun tube, elevation of the gun tube, odometer readings, fuel levels, and ammunition levels.

UPAS data displays were used to assess how well the actions of individual entities were controlled by mission, enemy, time, terrain, and troop variables. These displays were also used to assess how well entities worked together as part of armor or mechanized infantry platoon-level organizations in performing collective tasks described in the Mission Training Plan (MTP) document appropriate to each type of unit. Behavioral assessments in the individual entity mode included examinations of the effects of terrain on vehicle speed, rate of fires when confronted with target arrays, land navigation, and scanning behavior. Group behaviors assessed at platoon level included quality of formations, smoothness of transitions among formations, observation (scanning) within assigned sectors, reaction to contact, and the conduct of an assault.

This paper reviews selected types of measures of performance (MOPs) for which the UPAS is well suited. For a comprehensive report of the results of the comparison study, see the ModSAF 1.2/SAF 4.3.3 Comparison Study Summary Report (Loral, 1994).

3. Results

3.1 River Crossing

One of the first variables assessed at the entity level was the ability of an entity to locate and cross fordable water. Both ModSAF and SAFOR vehicles had difficulty with this task. For ModSAF vehicles in particular, the availability of a bridge or fordable area appeared to be little help. Figure 1, a UPAS Battle Flow display, shows the movement of a ModSAF vehicle during its approach to a river. Confusion and backtracking behavior are evident even in the presence of a bridge. Although SAFOR vehicles crossed bridges without much difficulty, the behavior of both SAFOR and ModSAF vehicles resembled that of the vehicle shown in Figure 1 when bridges were absent.

3.2 Terrain Slope and Speed Variation

The UPAS data tables proved effective in assessing speed adjustments associated with changes in the slope of terrain. For example, Table 1, from the

Speed (Km/Hr)	% of Time Moving at This Speed	Speed (Km/Hr)	% of Time Moving at This Speed
1	1%	26	1%
3	1%	27	1%
8	1%	28	1%
9	1%	29	1%
10	1%	30	1%
12	1%	32	1%
13	1%	33	1%
14	2%	35	2%
15	1%	36	54%
16	6%	37	2%
17	2%	38	1%
18	1%	39	1%
19	1%	50	1%
20	1%	62	1%
22	1%	73	1%
24	1%	76	1%
25	2%		

Table 1. % of Time ModSAF Tanks Spent Traveling at Various Speeds.

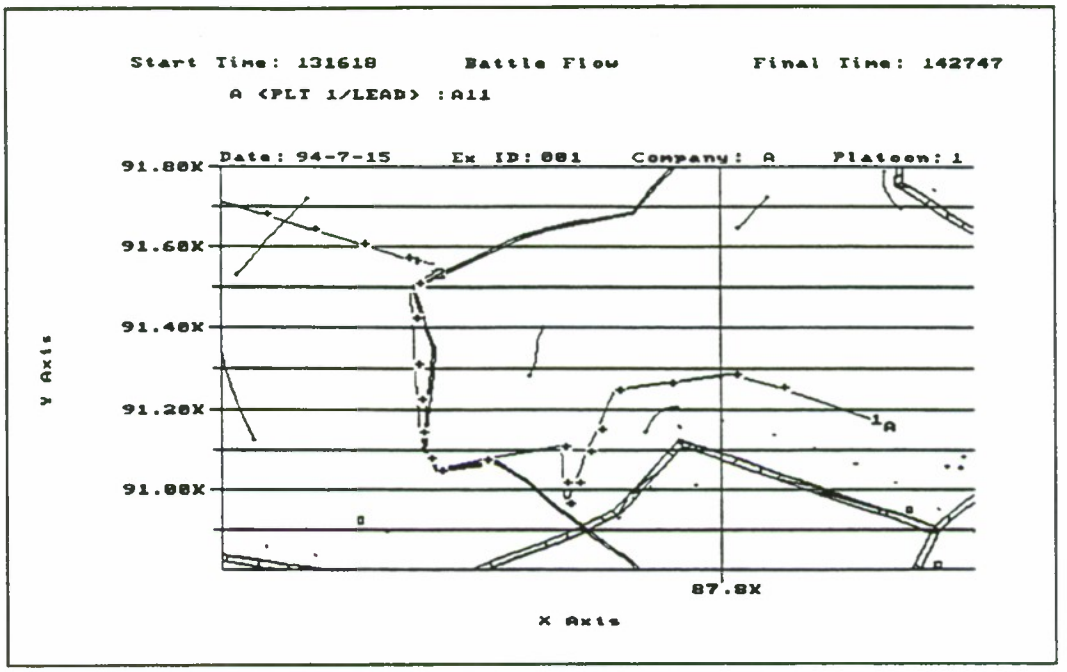


Figure 1. Trace of Movement of ModSAF Vehicle Showing Confusion and Backtracking as River is Approached.

UPAS GPLT table, shows the percentage of time a ModSAF entity spent traveling at various speeds. For the exercise segment captured in this table, the vehicle encountered inclines and declines of as much as 40 percent. The fact that slightly more than half of the terrain was approximately level is reflected byt he fact that 54 percent of the travel time was spent at 36 Km/Hr. In contrast, a SAFOR vehicle traveling over the same path spent 98 percent of the time at a speed of 29 Km/Hr demonstrating minimal responsiveness to terrain variation.

3.3 Firing Variables

During the comparison study, a "turkey shoot" scenario was employed to systematically compare firing accuracies across multiple ranges. All targets were stationary and non-firing. During the first set of turkey shoot scenarios, there was only one firing M1 tank, and it was given unlimited ammunition. This firing entity was given a "reasonable" amount of time to fire on the target entities. The original intent of the exercise was to allow the firing entity to continue firing until all of the target entities were destroyed. While, in theory, this would have provided a more robust comparison, it turned out to be impractical because, in several situations, the ModSAF firing entities required far more rounds than expected to inflict catastrophic kills on the targets. In these cases, the exercises were halted when the rate of fire slowed or when it appeared that the effectiveness of continued firing had decreased to near zero.

For the first of these scenarios, a company of 10 T-72 vehicles was placed at a range of 750 meters, and the firing competence was set to 1.00. After the firing was complete, a new target company was placed at 1500 meters and then at 2500 meters. These three firing scenarios were conducted with both ModSAF and SAFOR as the firing entity. The data for the turkey shoot scenarios are presented in Table 2.

From the data presented in Table 2, two conclusions are clear. First, the rate of fire for SAFOR entities was considerably faster than that of the ModSAF entities. In interpreting the SAFOR rate of fire data however, it should be noted that the SAFOR entities showed some peculiar firing

Range	CGF	Hits	Misses	Mean Time Between Rounds
750	ModSAF	19	1	11
	SAFOR	15	0	6
1500	ModSAF	17	5	10
	SAFOR	12	0	6
2500	ModSAF	22	18	10
	SAFOR	18	1	6

Table 2. Number of Hits and Misses and Mean Time Between Rounds for MODSAF and SAFOR Tanks Firing at Stationary Targets at Three Ranges.

behavior. While they were firing, there was very little deviation from the mean time between rounds of six seconds. However, they tended to stop firing while targets were still available. These breaks in firing were relatively long (as much as two minutes) and were subsequently removed from the rate of fire data. On several occasions, SAFOR entities that had stopped firing had to be "prompted" to evoke more firing. Usually, this prompting was done by setting the target entities in motion.

The second conclusion to be drawn from the data in Table 2, is that fire coming from SAFOR entities was unrealistically effective. If they fired, they almost always hit the target. This was evident throughout the exercises and will be shown in most of the firing data presented in this paper.

Bradley vehicles were also tested in the turkey shoot design. ModSAF and SAFOR fires appeared to be highly accurate across all ranges for both TOW and 25mm fires. Table 3 presents the accuracy data for TOW and 25mm fires by range and CGF type.

Range	CGF	Ammunition	Hits	Misses
750	ModSAF	TOW	33	4
		25 MM	174	4
	SAFOR	TOW	24	0
		25 MM	27	0
1500	ModSAF	TOW	11	0
		25MM	170	26
	SAFOR	TOW	24	0
		25 MM	188	0
2500	ModSAF	TOW	21	1
		25 MM	48	48
	SAF	TOW	60	2
		25 MM	90	1

Table 3. ModSAF and SAFOR Bradley Hits and Misses as a Function of Ammunition and Range.

It is of special interest to note that 25mm fires from both ModSAF and SAFOR Bradleys inappropriately resulted in catastrophic kills of tanks. Although the probability of kills appeared to drop with increased range, catastrophic kills resulted from 25mm fire even at 2500 meters. Problems with the effectiveness of 25mm fire were further demonstrated when a ModSAF Bradley platoon was later placed in a defensive position. Both 25mm and USSR 30mm rounds were found to hit and kill targets at impossible ranges. Hits were recorded at ranges as great as 3,452 meters for 25mm fire and 3,390 meters for 30mm fire. The longest catastrophic kill occurred at 3,350 meters with 25mm fire. Overall, weapons effects for 25mm rounds were unrealistic.

TOW fires, on the other hand, appeared to be less effective than expected. For example, in the turkey shoot scenario, a SAFOR Bradley fired 60 TOW missiles to achieve 10 catastrophic kills on tanks at 2500 meters (see Table 3).

Bradley firing effectiveness was also assessed while firing vehicles were on the move. A stationary, non-firing, T-72 platoon provided targets. The overall effectiveness of fire for both TOW and

25mm rounds appeared more realistic under these conditions. Table 4 shows the combined results of firing from four individually tested SAFOR Bradleys.

AMMO	RESULT	NUMBER
TOW	HIT	9
	KILL	6
	MISS	1
25 MM	HIT	70
	KILL	0
	MISS	1

Table 4. TOW and 25 MM Engagements for SAFOR Bradley Platoon.

Only one ModSAF Bradley was observed in the same situation. The results are presented in Table 5.

AMMO	RESULT	NUMBER
TOW	HIT	3
	KILL	6
	MISS	1
25 MM	HIT	70
	KILL	0
	MISS	1

Table 5. TOW and 25 MM Engagements for ModSAF Bradley Platoon.

3.4 Damage Sustained

The damage sustained while under fire was assessed for both ModSAF and SAFOR entities. In the first exercise to assess damage sustained, a platoon of red ModSAF tanks fired on a platoon of non-firing blue ModSAF tanks at a range of 1500 meters. The red tanks fired 120 HEAT and SABOT rounds and recorded only one catastrophic kill. All of the blue tanks received fire power and mobility kills very early in the exercise but later rounds appeared to have little effect. The one entity that became a catastrophic kill did so only after being hit by 18 rounds. These findings suggest that there were either problems associated with the vulnerability of the blue ModSAF M1s or with the firing effectiveness of the red ModSAF entities. When red SAFOR entities fired on blue SAFOR M1s in the same format, only 12 rounds were required to inflict catastrophic kills on the entire blue platoon.

A turkey shoot scenario was then conducted to assess damage sustained. Both ModSAF and SAFOR entities were fired upon by a single blue SAFOR vehicle. In these scenarios, the most hits received by any entity before becoming a catastrophic kill was seven. This was a ModSAF entity at 750 meters. The number of rounds required to inflict catastrophic kills are presented in Table 6 by range and SAFOR type.

	Rounds Required to Kill Target			
	1	2	3	4 or more
ModSAF				
750	7	1	1	1
1500	2	2	1	1
2500	5	1	0	2
SAFOR				
750	6	2	1	0
1500	8	2	0	0
2500	5	2	1	1

Table 6. Number of Hits by SAFOR Tank Rounds Before Targets Became Catastrophic Kills as a Function of Range and CGF Target Type.

Overall, there were no major differences between the ModSAF and SAFOR target entities during the turkey shoot scenarios. These findings suggest that the reduced vulnerability of ModSAF entities noted above was due to a problem in the firing effectiveness of ModSAF.

3.5 Movement in Formation

The UPAS provides a wide variety of position and movement related displays which can be used to assess the quality of formations. Several, but not all, of these displays are presented along with brief descriptions of the specific ways they were used in the ModSAF/SAFOR comparison study. A more robust analysis of ModSAF and SAFOR movement data is presented in the ModSAF 1.2/SAFOR 4.3.3 Comparison Study Summary Report (Loral, 1994).

On numerous occasions, formations were judged on the level of "perfectness" exhibited during movement. The "goal" of the CGF was to execute formations in a manner that was not too perfect (e.g. perfect an invariable spacing among entities) and yet not involving fixed, uncontrollable errors. In the majority of cases, variation within formations appeared to be fairly realistic with distances between entities varying over time. That is, they were not too "perfect." During a road march, for instance, the accordion behavior typical of manned units was observed. The UPAS Battle Snapshots in Figures 2 and 3 show a ModSAF platoon in exactly this situation.

A plot of the distance of each of three vehicles from a fourth over time can also be generated with the UPAS. This plot was used to identify points in time when major changes occurred in distances among entities. An example of this graph is shown in Figure 4. Notice that the time represented in Figure 4 includes the times at which the Battle Snapshots in Figures 2 and 3 were taken.

The corresponding displays for SAFOR platoon movement during the same road march showed similar variability. On several occasions, SAFOR vehicles had unique difficulties in maintaining an orderly line formation. For example, The UPAS Battle Snapshot in Figure 5 shows SAFOR vehicles passing one another during the road march. This is an example of one of the fixed uncontrollable errors we want to identify for removal.

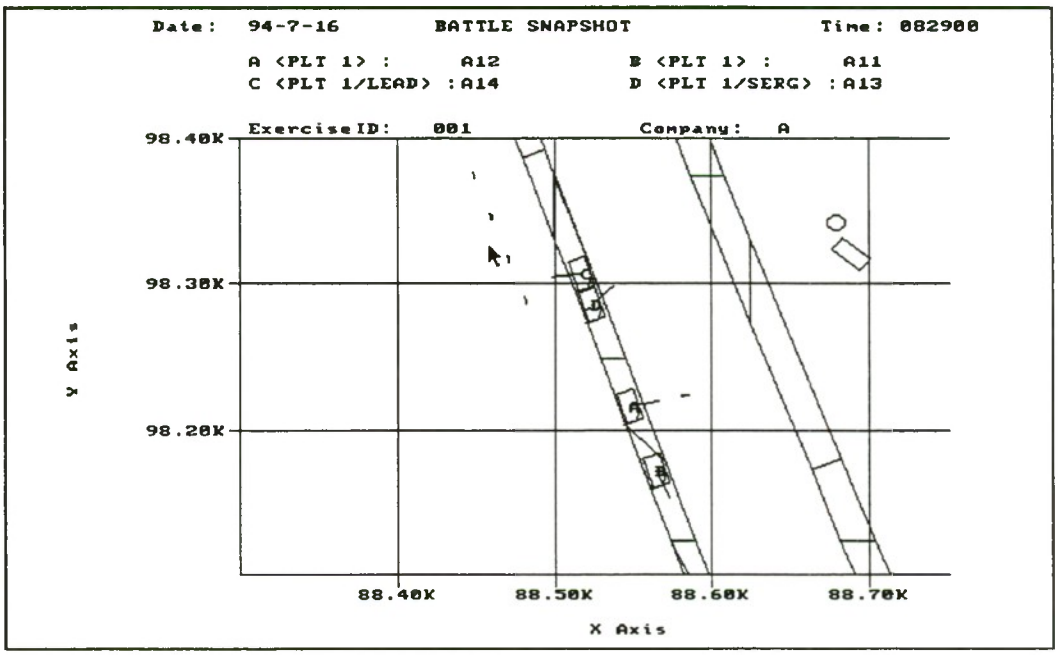


Figure 2. Bunching up of ModSAF Tanks During Movement.

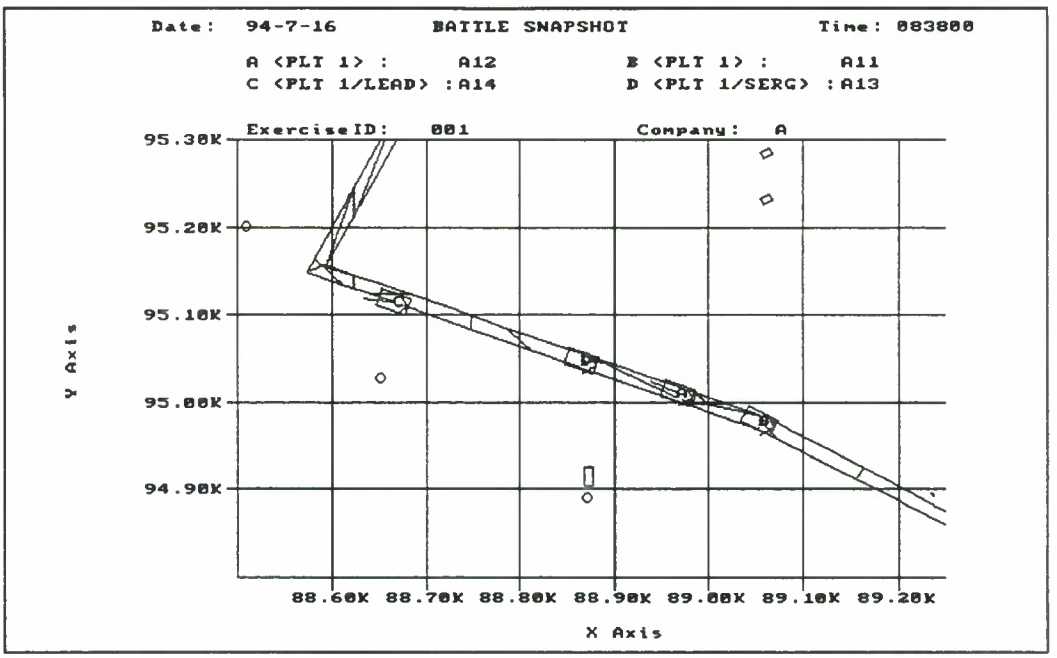


Figure 3. Gap in ModSAF Platoon Formation During Movement.

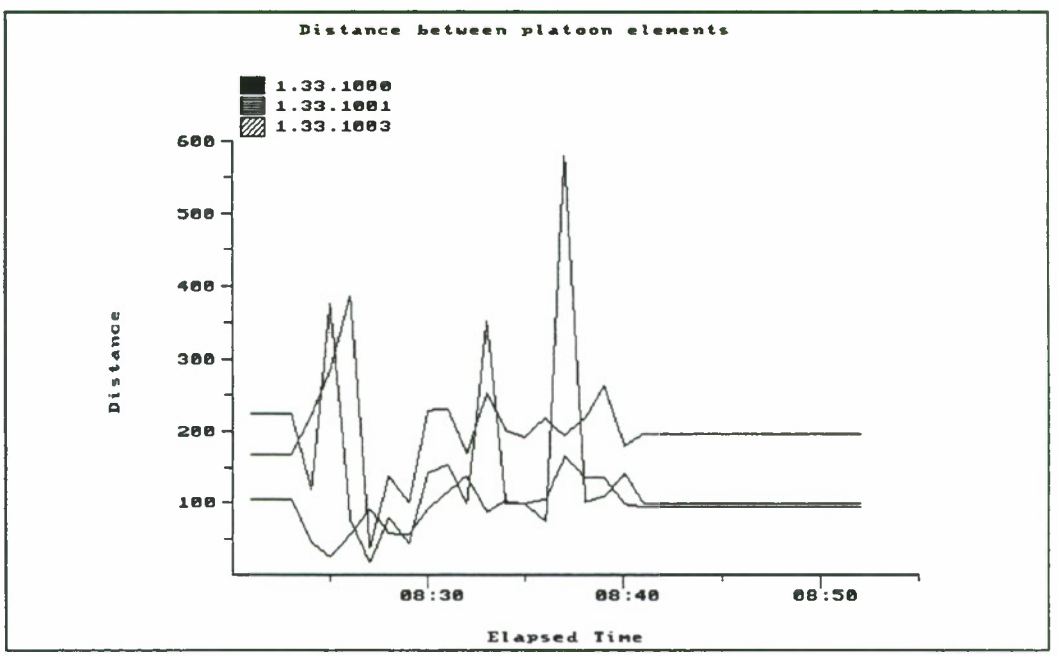


Figure 4. Graph Showing Distance Over Time Between Each of Three ModSAF Tanks and the Fourth Tank in a Platoon During a Tactical Road March.

3.6 Changing Formation

Both ModSAF and SAFOR demonstrated similar problems when changing from one formation to another. The most common problem was the exposing of vehicle flanks to the enemy during transitions. This occurred because the adjustments from one formation to the next were made by

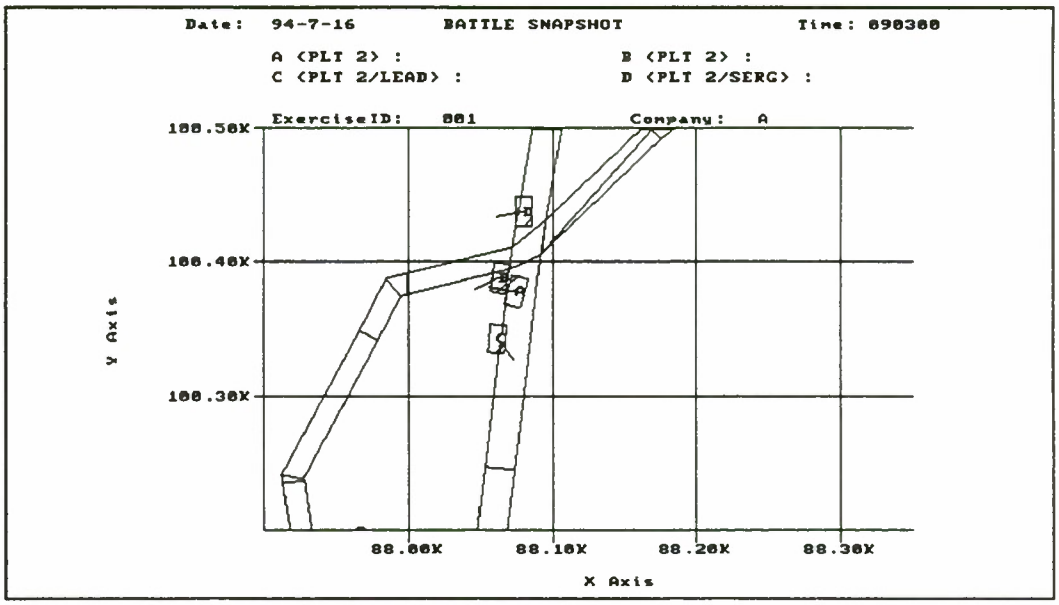


Figure 5. SAFOR Entities Passing One Another During Tactical Road March.

taking abrupt, sharp turns without regard for terrain or enemy position. The UPAS Battle Snapshots in Figures 6 and 7 show this problem clearly. Figure 6 shows a ModSAF platoon in a wedge formation. Figure 7 shows the same platoon approximately 20 seconds later in the middle of the transition from the wedge to a line formation.

3.7 Cross Country

Numerous problems resulted from encounters with natural obstacles for both ModSAF and SAFOR platoons during cross country marches. Canopied areas and tree lines proved difficult for both CGF systems to navigate. For example, tree lines invariably interrupted CGF platoon formations.

The UPAS Battle Snapshots shown in Figures 8 and 9 depict a SAFOR platoon approaching and then navigating a tree line. In this case, and many others, the integrity of the platoon was reduced when the two sections lost line-of-sight (LOS). In some cases, a single tank split from the others to avoid the tree line. There was no clear logic behind which entities were able to pass through the tree lines and which entities had to circumvent them as impenetrable obstacles.

3.8 Scanning Behavior

Numerous observations of inadequate scanning behavior were made throughout the exercises. One goal was to assess whether the entities provided good security by collectively covering all of the critical sectors. These sectors differed as a function of the formations used and other aspects of the tactical situation. The UPAS Battle Snapshot display captures turret orientation, and an example of inadequate scanning behavior can be seen in the Battle Snapshot in Figure 10.

The UPAS data tables can also be used to calculate slue rates and ranges. For example, scanning behavior for single ModSAF and SAFOR Bradley vehicles was assessed during a cross country march to a halt. In this situation, each entity should be providing 360 security. Data from the UPAS indicated that the ModSAF Bradley scanned a full 360 degrees in a time of one minute and thirty-five seconds or at a rate of 67 mils per second. The SAFOR Bradley scanned a small sector to the left front that covered just 589 mils. The rate of scanning for the SAFOR entity was not calculated as the turret azimuth often remained constant for many seconds. Unfortunately, the pattern remained the same when ModSAF and SAFOR entities move as part of a platoon. That is, formation variables had little effect on scanning.

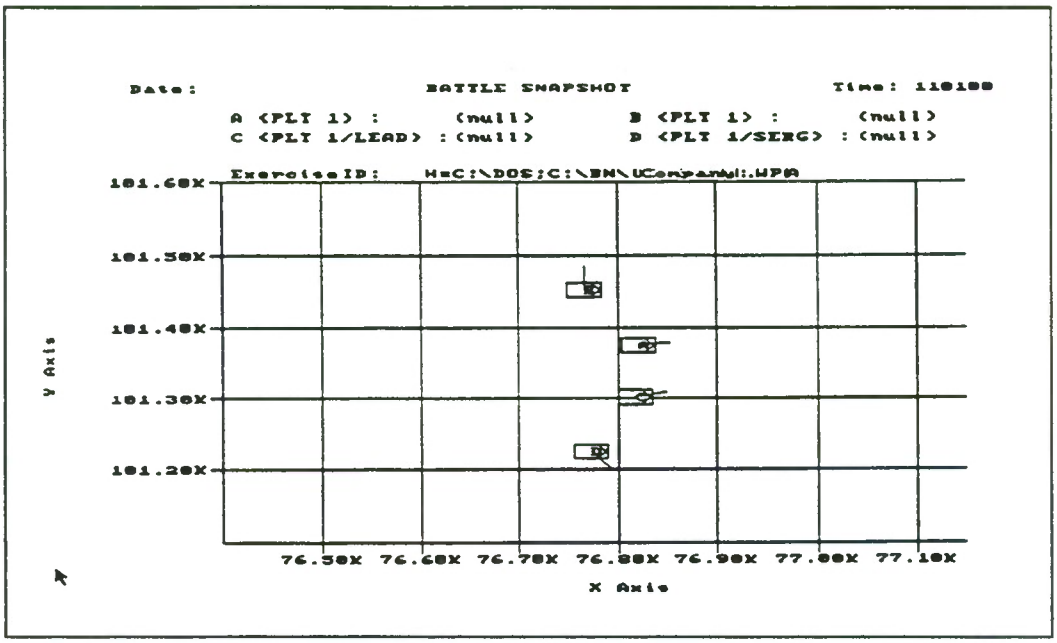


Figure 6. ModSAF Platoon Moving in the Wedge Formation.

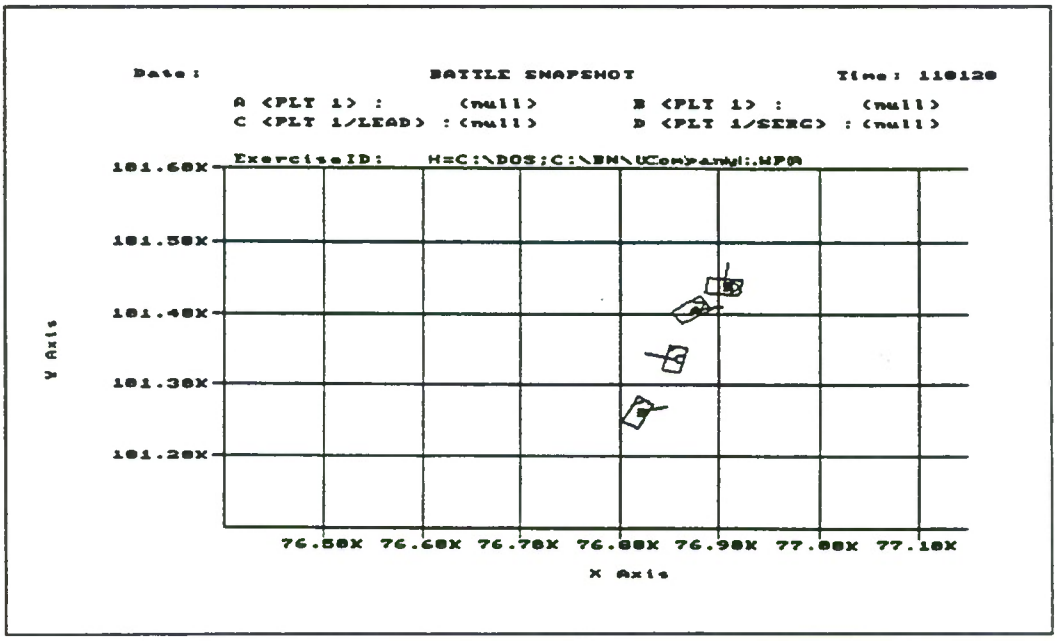


Figure 7. Tanks Expose Their Flanks to the Enemy as the Platoon Changes Formation 20 Seconds Later.

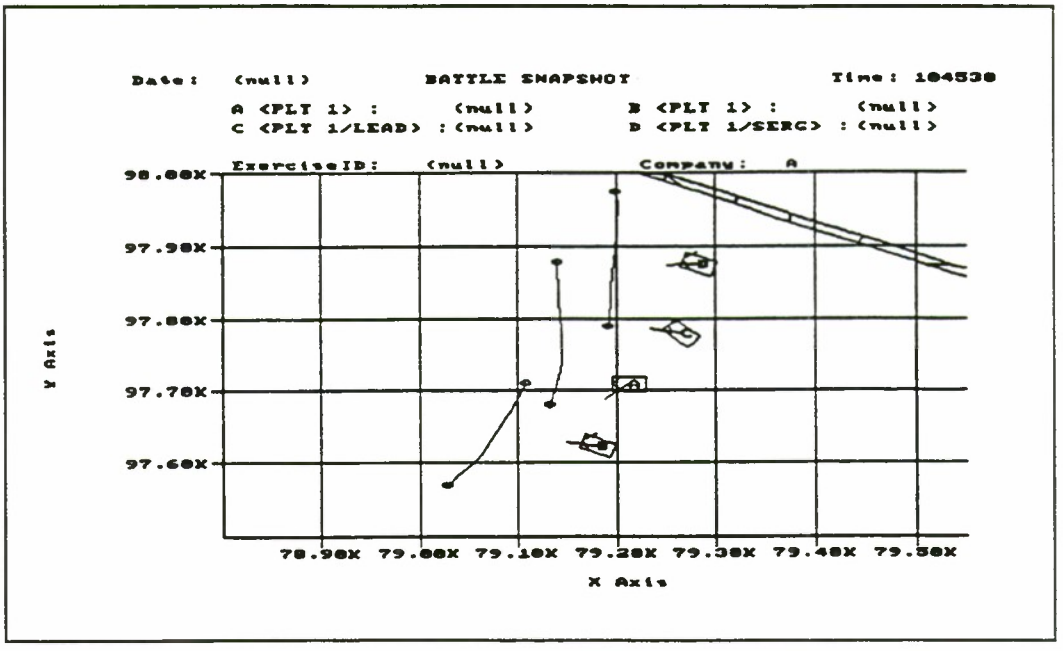


Figure 8. SAFOR Platoon Moving in the Line Formation as it Approaches a Tree Line.

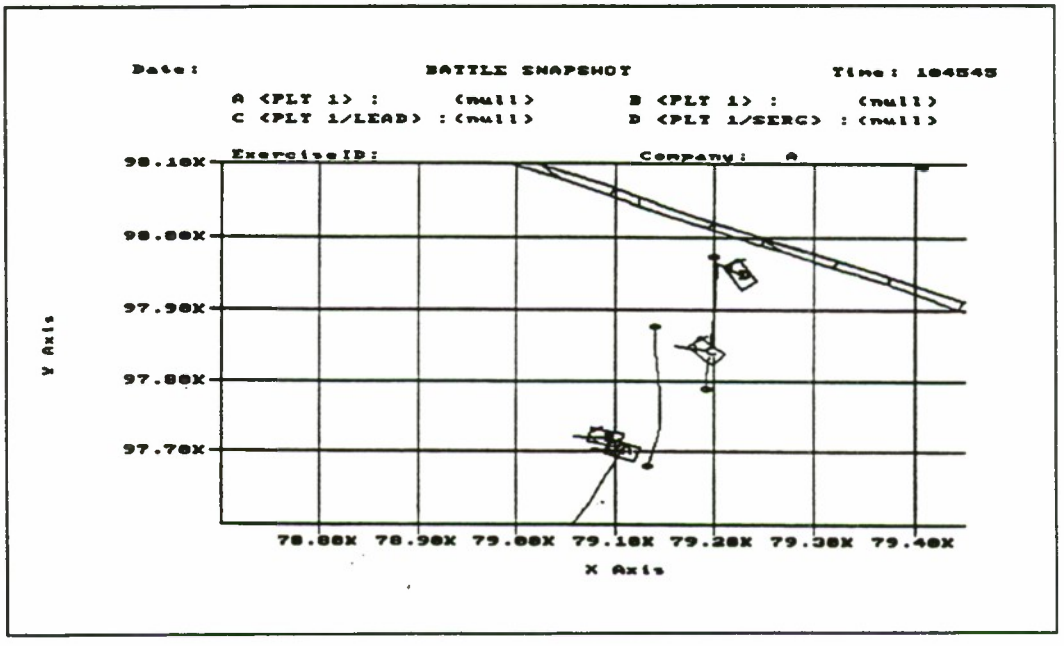


Figure 9. Formation is Disrupted by the Treeline.

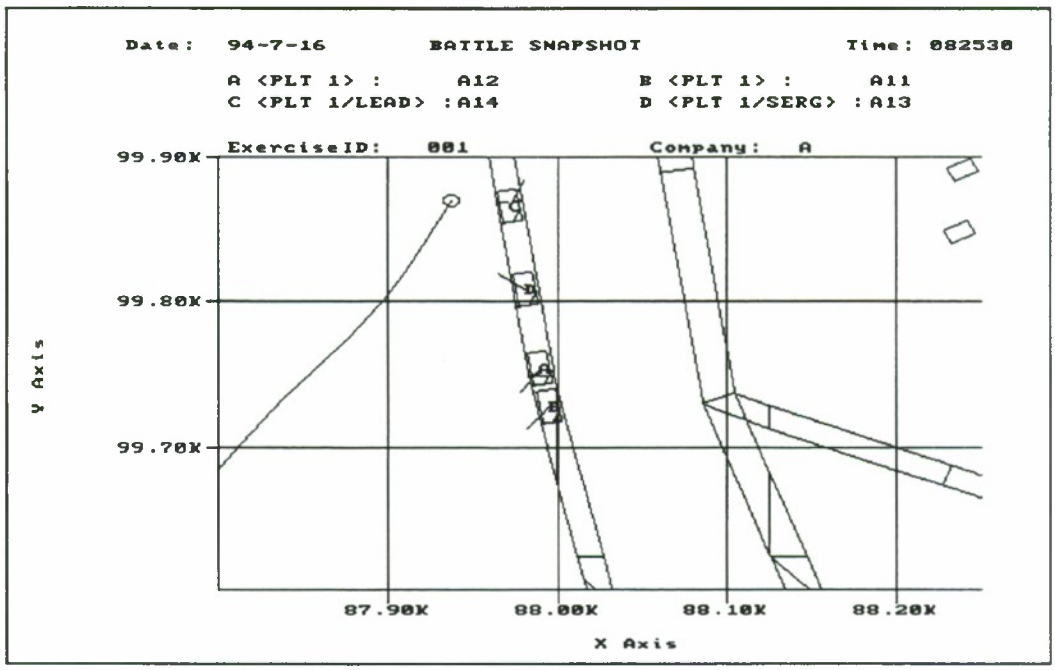


Figure 10. Gaps in Sectors Covered By Scanning of MODSAF Tank Platoon During Tactical Road March.

3.9 Reaction to Contact

ModSAF and SAFOR entities (M1's, M2's, and Bradleys) were assessed for reactions to contact during road and cross country marches on numerous occasions. In general, entities reacted to contact by firing only. That is, they did not change speed or formation, and never tried to use cover and concealment or deploy against the threat.

SAFOR entities exhibited the same unrealistic firing accuracy indicated earlier. On one occasion, a SAFOR tank platoon on a road march fired three rounds and destroyed three T-72s, that were set up for an ambush, at a range of 1500 meters. The SAFOR entities never slowed down and there was no adjustment in their gun tube orientation.

ModSAF entities were less accurate in their firing but their overall reaction to contact was similar to that of the SAFOR entities. ModSAF tank platoons varied their speed only because of changes in terrain elevation and their direction of travel never varied. If approaching the enemy head on upon contact, the approach was not altered. Most entities participated in the firing and there appeared to be some coordinated firing within platoon sections. This conclusion was based on the correlation of fire times for entities within sections. There did not appear to be any coordinated fire between platoon sections. Further, in several company level exercises, there was no evidence of coordination (firing or otherwise) between platoons except that they all moved in the same general direction

3.10 Conduct of Assault

Behaviors associated with the conduct of assaults were generally the same as those observed during reaction to contact. That is, both SAFOR and ModSAF entities maintained constant speeds (except that ModSAF adjusted speed for changes in terrain elevation), did not change formation, and failed to use cover and concealment. SAFOR fire was extremely lethal as usual. During one assault, a SAFOR tank platoon, travelling at a constant speed of 40Km/Hr, opened fire on an enemy platoon at a range of 2350 meters. Each entity fired at least once, however, only five total rounds were fired. All five rounds resulted in hits, and the enemy platoon was destroyed in seven seconds.

4. Conclusion

Overall, both types of CGF displayed inadequate sensitivity to the mission, enemy, time, terrain, and troop (METT-T) variables that should be controlling CGF behavior. On certain measures, differences were observed between the CGF.

Illustrative examples of the results are as follows:

o one type of CGF was superior to the other in terms of the ability to adjust individual or group behavior to fit the terrain situation, but the behavior of both types of CGF could be disrupted by certain types of terrain

o both versions make hard, abrupt turns when changing formations causing flanks to be exposed to enemy positions

o one version demonstrated little or no movement of gun tubes (scanning) during movement while the other version tended to scan aggressively, but the coverage of sectors by the platoon for the second version failed to meet MTP standards

o both versions used inappropriate assault techniques, and vehicle speeds were not properly adjusted during an assault

o rate and effectiveness of fire was much higher for one version than the other

o many cases were observed where firing effectiveness or vulnerability of entities to fires were at unrealistic levels

o effectiveness of fires varied across weapon systems within a version of CGF as well as differing between CGF versions

5. References

Loral (1994). *ModSAF 1.2/SAF 4.3.3 Comparison Study Summary Report* (ADST/WDL/TR--94-W003328). Orlando, FL: Loral ADST Program Office.

Meliza, L.L., Bessemer, D.W., & Tan, S.C (1994). *Unit Performance Assessment System Development.* (ARI Technical Report 1008). Alexandria, VA: U.S. Army Research Institute for the Behavioral and Social Sciences.

Mengel, L.L. (1994) *ModSAF Summer Exercise (SUMEX-I) Final Report.*

Vaden, E.A., Meliza, L.L. & Johnson, W.R. (1994). "Using the Unit Performance Assessment System (UPAS) to Measure Modular Semi-Automated Force Behavior". *In Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation.* Orlando, FL: University of Central Florida Institute for Simulation and Training.

6. Biographies

Larry Meliza is a Research Psychologist with the U.S. Army Research Institute Simulator Systems Research Unit at Orlando, FL. He earned his Ph.D. degree in psychology from the University of Arizona. His experience in collective performance measurement includes measuring the effects of tactical engagement simulation on collective training, developing prototype Army Mission Training Plan documents to guide collective training, and developing/using the Unit Performance Assessment System (UPAS) for training research.

Eric Vaden is a doctoral student in Human Factors Psychology at the University of Central Florida. His primary area of study is human computer interaction and he spent a four month internship working with the usability group at Microsoft Corporation. When this work was performed he was a Research Fellow with the Consortium Research Fellows Program at the U.S. Army Research Institute Simulator Systems Research Unit at Orlando, FL. His experience in the collective performance arena includes using the Unit Performance Assessment System for training research and CGF assessment, and evaluating the human-computer interfaces of several After Action Review (AAR) systems.

The Use of Automated Regression and VVA Testing in ModSAF

Paul Monday, James Perneski
Loral Advanced Distributed Simulation
50 Moulton St.,Cambridge, MA 02138
monday@knox.loral.com
jpernesk@camb-lads.loral.com

1. Abstract

This paper discusses the reasons for and the implementation of automated testing techniques in the ModSAF program. There are, in general, two major areas to be tested in ModSAF. The first is the overall correctness of simulated behavior; since this area is tested whenever code changes are submitted, this type of testing is called Regression testing. The second area of testing is that of adherence to specific model criteria, which is called VVA. Each of these areas requires different testing and evaluation techniques and will be discussed separately. The tools used in the implementation will be briefly described, and the benefits in terms of manpower and time will be discussed. In addition, ongoing and future development will be discussed.

2. The ModSAF System

The ModSAF software architecture is an extensible set of reusable software modules which allows rapid development and testing of new agents in the DIS simulated environment. The ModSAF application program uses the ModSAF architecture to construct a Semi-Automated Forces (SAF) system which is currently used for a variety of different applications including a DIS test bed used by researchers implementing intelligent control algorithms, a SAF system supporting training exercise for the National Guard using manned simulators, an architectural prototype for SAF design for the CCTT program and as the SAF system to be used to support combat development experiments for the BDS-D program, including the Anti-Armor Advanced Technology Demonstration (A2ATD) and Line of Sight Anti-Tank Program (LOSAT).

ModSAF supports DIS 1.0, 2.0.3 and SIMNET protocols ; it can be run in a variety of configurations using one or multiple workstations. It is supported on SGI, Mips, SUN Sparc, and IBM Risc platforms.

It is a large program, consisting of over 500,000 lines of executable C code distributed over 300 libraries

3. Need for Automatic Testing

Since ModSAF is used for rapid prototyping and development, there are a large number of contributors of code to the program, including contract contributors outside of the Loral staff. At any time in the ModSAF release cycle, there are several projects underway, which are being developed independently, and therefore must be integrated.

Code integrations are performed at least on a weekly basis during the development cycle to accommodate these large and rapid contributions. However each contribution must be validated before a new contribution is added to the program. To accomplish this validation, a suite of tests must be performed to insure that the basic functionality of the program existing before the integration is still intact. Before each release, a more extensive period of testing takes place in order to validate if the behavior of the modeling, such as Direct Fire Damage Assessment continues to perform as originally specified. This pre-release testing also exercises the entire program to insure that it is properly working.

The basic method used in most of the testing described above is to reload and run "Scenario" files created in previous releases and integrations, and to observe if they continue to perform as before. A Scenario is a file that captures all of the information concerning the entities, their positions, their assigned tasks and the terrain on which they are located. A Scenario therefore captures a moment in time and space from which the exercise can continue. A tester then observes the subsequent behavior. This method is expensive in both time and manpower; the time it takes to run the scenario and the person necessary to observe the test and determine if it passed.

Improvements to the testing procedures take place by automating the execution of the scenarios and recording the results in such a way that deviations from expected behavior are highlighted. VVA testing, as explained later, is able to be more objective because many of its procedures involve gathering data, such as hits upon a target, for subsequent data analysis.

4. Regression Testing

As discussed above, a suite of tests are performed after each integration of new code into the ModSAF program, and a larger suite of tests are performed before each release. The tests consist of reloading, running, and observing a standard set of scenarios. A large part of these tests have been automated as described below.

4.1. Present Implementation

A procedure has been developed to cycle through a directory of scenario files. For each to the scenarios, the procedure will start a debugger program, and from within the debugger, the ModSAF program itself will be started. The scenario is then loaded, and executed.While executing, some information is recorded, most notable information concerning successful loading and the code location of program crashes, if they occur. If a crash does occur, the procedure records where in the code the crash occurred, exits the debugging program and repeats the procedure for the next scenario in the directory. On the other hand, if the scenario has executed for a time, determined as an argument to the procedure, the ModSAF program is terminated, the debugging program is terminated, and the procedure is repeated for the next scenario in the directory.

This procedure is written as a shell script and a few points are worth discussing. The scenarios used are designed to create and simulate all the entities and units of entities which ModSAF can create. As new entities are added, new scenarios are added, increasing the amount of testing. The successful loading of the scenarios demonstrates that conversion files, which are generated to update the data structures of entities, work properly. Since there are many scenarios to be tested, a time limit for each must be imposed. New scenarios built for automated testing must execute the questioned behavior with a time limit. A debugging program is used not only to prevent time consuming core dumps, but to aid the responsible programmer with his analysis.

Using the above procedure, typically 170 scenarios are loaded and executed, and the procedure will run continually for over 10 hours. Before an integration is finally accepted, this procedure is run overnight and the recording file scanned for any anomalies.

4.2 Ongoing and Future Work

There are two primary directions of extending the above procedure in order to make it more widely applicable. Both of these extensions, described below, are presently undergoing development and implementation.

4.2.1 Multiple Machine Implementation

The above procedure is designed to be executed on one workstation running ModSAF as a pocket system, that is the work station acts as both the user interface to ModSAF and also simulates all entities. However, the "natural" environment for simulation exercises is spread over several work stations, some of which act as user interfaces to the exercise and others do the work of simulating the entities and events. Of course they are all communicating over a network.

The testing of scenarios should also be done automatically in the distributed environment. For example, it is important that, if one of the simulators goes down during an exercise, the entities being simulated there are taken over by other simulators.

In order to test scenarios automatically over a network, a client- server model is being developed. This will establish another level of communication between the workstations above the communications that ModSAF needs. One machine will act as the server, which will command the registered clients to start the debugging program and to load ModSAF using the appropriate terrain necessary for the scenario to be loaded. The server will execute the scenario, and each client will record data locally during the execution of the scenario. The server can instruct one or more simulators to stop running to determine it the simulated entities are passed to the remaining simulators. Finally, the server can instruct the clients to stop ModSAF and the debugging program in order to repeat the cycle for the next scenario.

4.2.2 Seeking Significant Events

Another direction of extension being pursued is the recording the occurrence, or non occurrence of specific events during the execution of a scenario. All the events, such as collisions between vehicles and or between vehicles and the environment are sent as network packets. Presently these packets can be recorded for analysis by any of several data logging programs. However using a logger in this way, more data than necessary is recorded, and must be analyzed at later time.

Code will be developed to allow the procedures above to look for events that indicate if expected behavior was executed. For example, the procedure could be set to look for and record collisions and positions while executing a scenario of a unit of vehicles crossing a bridge. If collisions are recorded, the test fails. This will provide a pass-fail criterion with out any further analysis.

5. Verification and Validation Testing

Since ModSAF is becoming an accepted model for combat developments activities, the Validation and Verification (V&V) of important models in ModSAF has become an essential part of the software development process. V&V tests should be performed when these models are developed and when they are changed. In addition, mini-V&V check tests should be performed on each ModSAF release to ensure that none of the important models was inadvertently changed.

Several models have been evaluated for V&V by the Army Material Systems Analysis Activity (AMSAA), including target acquisition (LibVisual), direct fire delivery accuracy (LibBalGun), direct fire rate-of-fire (LibBalGun), direct fire vdamage assessment (LibDfDam), and indirect fire damage assessment (LibIfDam). Specific tests and data requirements were specified by AMSAA for each model. A data structure that can be transmitted via a DIS Event Report PDU was developed for each model that contains all of the detailed, internal parameters necessary for the evaluation.

5.1 Target Acquisition

The V&V test for target acquisition is intended to examine table lookups of parameters like target contrast, calculation of values like critical dimension, and evaluation of output parameters including p_infinity. Data contained in the VVA data structure includes:

target entity id
sensor type (optical, infra-red)
exposure (full, hull-defilade)
magnification
critical dimension
intervisibility
range
apparent contrast
cycles on target
acquisition time
p_infinities for four acquisition levels
random numbers used

A V&V check test can be performed by running a pre-defined scenario, and recording the resulting PDUs. VVA data from the appropriate Event Report PDUs is extracted and analyzed. Parameters like range and intervisibility are independently calculated by the ADST Data Collection and Analysis (DCA) system for comparison. The table lookups and calculations for the other parameters are duplicated in the DCA system. Any deviations are reported.

5.2 DF Delivery Accuracy

The V&V test for direct fire delivery accuracy is intended to examine table lookups of parameters like the biases and dispersions, the calculation of parameters like the miss distance, and the determination of hit or miss.

Data contained in the VVA data structure includes:

shot's event id
firer location
target location
aimpoint location
firer and target movement (SS, SM, MS, MM)
exposure (full, hull-defilade)
inputs biases/dispersions
output biases
horizontal and vertical miss distance
horizontal and vertical hit assessment

A V&V check test can be performed by configuring a ModSAF firer with a large amount of ammunition and with very small reload times. By presenting the firer with many targets in a variety of ranges, exposures, and aspects, a dataset containing several thousand shots can be acquired quickly.

VVA data from the appropriate Event Report PDUs is extracted and analyzed. Parameters like locations and firer/target movement conditions are independently calculated by the DCA system for comparison. The table lookups and calculations for the other parameters are duplicated in the DCA system. The random draws are statistically analyzed for reasonableness. Finally, the a statistical evaluation of the hit assessment algorithm is performed.

5.3 DF Damage Assessment

The V&V test for direct fire damage assessment is intended to examine calculation of lookup parameters like dispersion, table lookups of the Pks, transformation of the Pks, and damage assessment. Data contained in the VVA data structure includes:

shot's event id
target's location
impact location
exposure (full, hull-defilade)
range
dispersion (from center-of-mass)
aspect angle
elevation angle
looked-up Pks
only-type Pks
thermometer Pks
random number

indicated damage
before and after status

A V&V check test can be performed using a pre-recorded dataset that contains many direct fire impacts (Detonate PDUs), but has had the target of these impacts removed. Then, if a ModSAF vehicle is generated, and the pre-recorded dataset is played back, the new ModSAF vehicle receives the prerecorded impacts and performs damage assessment. In this manner, a test with several hundred impacts can be run in a couple of minutes.

As before, VVA data from the appropriate Event Report PDUs is extracted and analyzed. Parameters like range and dispersion are independently calculated by the DCA system for comparison. The table lookups, Pk conversions, and damage assessment are duplicated in the DCA system.

5.4 IF Damage Assessment

The V&V test for indirect fire damage assessment is intended to examine calculation of lookup parameters, table lookups of the lethal areas, calculation and transformation of the Pks, and damage assessment. Data contained in the VVA data structure includes:

shot's event id
target's location
impact location
impact-to-target range
cutoff range
damage function (cookie, carleton)
D0 (carleton)
angle of fall (carleton)
range/deflection miss distance (carleton)
firer-to-impact range (cookie)
slope, intercept (cookie)
pattern radius (cookie)
lethal areas
computed Pks
only-type Pks
thermometer Pks
random number
indicated damage
before and after status

A V&V check test can be performed similarly to the direct fire damage assessment test, except with an indirect fire munition. When a ModSAF vehicle is generated and the pre-recorded dataset is played back, the new ModSAF vehicle receives the prerecorded impacts and performs damage assessment. In this manner, a test with several hundred impacts can be run in a couple of minutes.

As before, VVA data from the appropriate Event Report PDUs is extracted and analyzed. Parameters like range are independently calculated by the DCA system for comparison. The calculation of the Pks by either method, Pk conversions, and damage assessment are duplicated in the DCA system.

6. Summary

It is necessary to perform a large number of tests to insure the quality of the ModSAF program as it is being developed. In this paper two types of testing where discussed, Automated regression testing and VVA testing.

The automated regression testing allows many tests to be preformed, and recorded. in such a manner that pass or failure is easily and quickly determined. The methods presently in place have been very successful, because they allow more scenarios to be executed and evaluated while using less time and requiring less supervision. These methods are being developed further.

VVA testing insures the quality of the ModSAF program by demonstrating conformance of a model to specifications. Methods have been developed here for the production and collection of data, and its analysis.

7 Acknowledgements

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058.

8. Biographies

Mr. Monday has worked on the SIMNET-D and ADST projects for BBN and Loral since 1987. He is currently Chief Analyst at the Mounted Warfare Test Bed (MWTB), Ft Knox, Kentucky where he designs and develops software for data analysis, ModSAF, and other simulations. He graduated from the University of Toledo in 1978 with a B.S. in geology and from Stanford University in 1979 with a M.S. in geophysics. Mr. Monday previously worked for 7 years in petroleum exploration.

Before joining Loral, James Perneski spent several years in the CAD-CAM industry; during which time geometric modeling was his main interest He received a B.S. from Lehigh University, and an M.A. from the University of Connecticut.

VERIFICATION AND VALIDATION OF MODULAR SEMI-AUTOMATED FORCES (ModSAF) IN SUPPORT OF A2 ATD EXPERIMENT 1

John G. Thomas
U.S. Army Materiel Systems Analysis Activity
Aberdeen Proving Ground, Maryland 21005-5071
jgthomas@arl.mil

1. Abstract

In support of the Anti-Armor Advanced Technology Demonstration (A2 ATD) program, the US Army Materiel Systems Analysis Activity (AMSAA) is responsible for the verification and validation of the physical models incorporated within the ModSAF model. The ModSAF verification, validation and accreditation effort is a joint effort between AMSAA and the US Army Training and Analysis Command (TRAC) - White Sands Missile Range (WSMR) with TRAC having the overall lead. TRAC is responsible for the verification and validation of the ModSAF combat behaviors.

ModSAF is a set of software modules and application programs that permits a single operator to control large numbers of vehicles on the virtual battlefield. ModSAF is being developed under the sponsorship of the U.S. Army Simulation, Training, and Instrumentation Command (STRICOM) and the Advanced Research Projects Agency (ARPA). The objective of ModSAF is threefold: 1) replace the current Simulation Network (SIMNET) Semi-Automated Forces (SAF) systems at the Battlefield Distributive Simulation-Developmental (BDS-D) sites, 2) support BDS-D experiments (A2 ATD, Horizontal Technology Integration, etc.), and 3) support ARPA programs (Synthetic Theater of War, etc.).

ModSAF was verified, validated, and accredited for A2 ATD experiment 1. The purpose of the first A2 ATD experiment was to validate virtual simulation (BDS-D) with live simulation (M1A2 Initial Operational Test and Evaluation) and to validate constructive simulation (ModSAF and CASTFOREM (Combined Arms Support and Task Force Evaluation Model)) with live and virtual simulation.

In support of ModSAF V&V, AMSAA conducted a series of ModSAF verification and validation check tests. In particular, for experiment 1, AMSAA conducted ModSAF check tests which focused on the following physical models and related data: Direct-Fire Vulnerability, Target Acquisition, Direct-Fire Delivery Accuracy, Direct-Fire Rate-of-Fire, Indirect-Fire Vulnerability, and Mobility. Similarly, TRAC-WSMR examined the behavioral algorithms for ModSAF. Since experiment 1 was strictly armor, the behaviors reviewed were all armored tactics.

Moreover, ModSAF performance was benchmarked in experiment 1 against CASTFOREM. Also, the employment of CASTFOREM to pre-experiment analysis provided the capability for refinement of input performance and scenario data prior to initiation of the experiment. CASTFOREM is a TRADOC (US Army Training and Doctrine Command) accredited, stochastic, constructive force-on-force combat simulation which has been employed in Army acquisition COEAs (Cost Operational Effectiveness Analysis) for years.

2. Introduction

The focus of this paper is to provide an overview of the ModSAF Verification and Validation (V&V) efforts in support of the A2 ATD program. ModSAF is the SAF (semi-automated forces) model currently used in A2 ATD. A2 ATD is a joint Department of the Army and Department of Defense program initiated with the goal of maturing Distributed Interactive Simulation (DIS) as a credible evaluation tool to support acquisition decisions. The purpose of the A2 ATD is to develop and demonstrate a verified, validated, accredited DIS capability to support anti-armor weapon system virtual prototyping, concept formulation, requirements definition, effectiveness evaluation, and mission area analysis on a combined arms battlefield at the Battalion Task Force or Brigade level.

The A2 ATD technical objectives are:

- Demonstrate DIS as an evaluation tool and verify, validate, and accredit simulators used in A2 ATD experiments, semi-automated forces, and the BDS-D simulation.

- Develop, demonstrate, and document techniques/analytical tools to analyze simulation results to include Verification and Validation of ModSAF.

- Demonstrate the linkage of constructive models (JANUS) to DIS.

- Demonstrate upgraded virtual prototypes (M1A2, M2A3/M3A3, NLOS, LOSAT) and virtual prototypes to be developed (AGS, JAVELIN, Comanche, EFOGM, Hunter).

3. ModSAF Model

ModSAF is a set of software modules and application programs that permits a single operator to control many vehicles on the virtual battlefield. The U.S. Army Simulation, Training, and Instrumentation Command (STRICOM) and the Advanced Research Projects Agency (ARPA) are sponsoring the development of ModSAF. The objective of ModSAF is threefold: 1) replace the current Simulation Network (SIMNET) Semi-Automated Forces (SAF) systems at the Battlefield Distributive Simulation-Developmental (BDS-D) sites, 2) support BDS-D experiments (A2 ATD, Horizontal Technology Integration, etc.), and 3) support ARPA programs (Synthetic Theater of War, etc.).

4. ModSAF V & V

In support of the A2 ATD program, the US Army Materiel Systems Analysis Activity (AMSAA) is responsible for the verification and validation of the physical models incorporated within the ModSAF model. The ModSAF verification, validation and accreditation effort is a joint effort between AMSAA and the US Army TRADOC Analysis Center (TRAC) - White Sands Missile Range (WSMR) with TRAC having the overall lead. TRAC is responsible for the verification and validation of the ModSAF combat behaviors.

Simulator and semi-automated forces VV&A and development of analytical tools to support the evaluation of causes of simulation outcomes were initiated in FY93 to provide the foundation for a series of six experiments. The first experiment, completed September 14, 1994, replicated two M1A2 Initial Operational Test and Evaluation (IOT&E) battles conducted at Ft. Hood during the autumn of 93.

ModSAF was verified, validated, and accredited for A2 ATD Experiment 1. The purpose of the first A2 ATD experiment was to validate virtual simulation(BDS-D) with live simulation (M1A2 Initial Operational Test and Evaluation) and to validate constructive simulation (ModSAF and CASTFOREM (Combined Arms Support and Task Force Evaluation Model)) with live and virtual simulation.

4.1 Physical Models

In support of ModSAF V&V, AMSAA conducted a series of ModSAF verification and validation check tests. In particular, for Experiment 1, AMSAA conducted ModSAF check tests that focused on the following physical models and related data: Direct-Fire Vulnerability, Target Acquisition, Direct-Fire Delivery Accuracy, Direct-Fire Rate-of -Fire, Indirect-Fire Vulnerability, and Mobility [Ref 3].

As one of the A2 ATD technical objectives, a set of DISATs (DIS Analytical Tools) to analyze simulation results to include V & V of ModSAF were developed. Moreover, the DISATs were used to support the analysis effort of A2 ATD experiment 1 to include V & V of the physical models employed by ModSAF.

ModSAF V&V exercises sometimes referred to as check tests, utilize the VV&A portion of the DISAT. At the time of execution of a ModSAF V&V exercise the ModSAF VV&A flags are set and the DISAT data logger is activated.

These flags activate the VV&A protocol data units (PDU's). These flags include the following [Ref 2]:

- Status Change VV&A Data
- Target Acquisition VV&A Data
- Direct-Fire Delivery Accuracy VV&A Data
- Direct-Fire Damage Assessment VV&A Data
- Indirect-Fire Damage Assessment VV&A Data

After the exercise is completed, the DISAT software used in conjunction with the logged file generates a series of computer files containing VV&A data tables. Figure 1 provides an overview of the ModSAF VV&A/DISAT operation.

For each VV&A flag set, multiple files are created except in the case of the status change flag. These files consist of target status change, direct-fire vulnerability, direct-fire delivery accuracy, target acquisition, and indirect-fire vulnerability VV&A data tables.

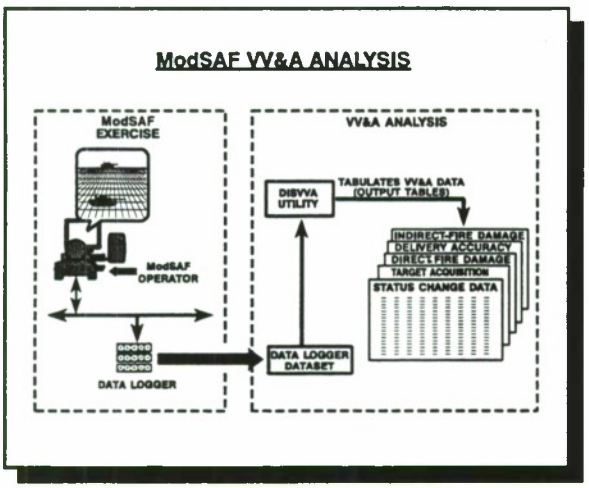


Figure 1. Overview of ModSAF VV&A Analysis

A summary of AMSAA's ModSAF VV&A efforts prior to A2 ATD Experiment 1 is provided in Figure 2.

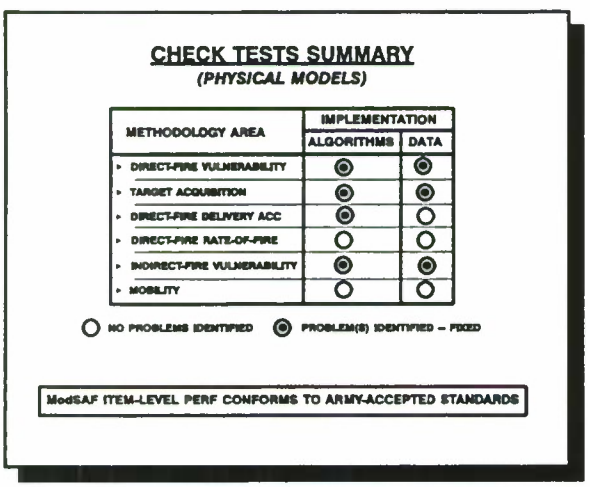


Figure 2. Check Tests V&V Summary

4.2 Combat Behaviors

Similarly, TRAC-WSMR examined the behavioral algorithms for ModSAF. Since experiment 1 was strictly armor, the behaviors reviewed were all armored tactics. Moreover, TRAC-WSMR identified behavioral algorithm deficiencies within ModSAF that will be addressed in future releases of ModSAF. Figure 3 shows 'work arounds' for these deficiencies [Ref 1]. For Experiment 1, manned intervention compensated for the behavioral deficiencies such that armor tactics and doctrine could be adequately represented.

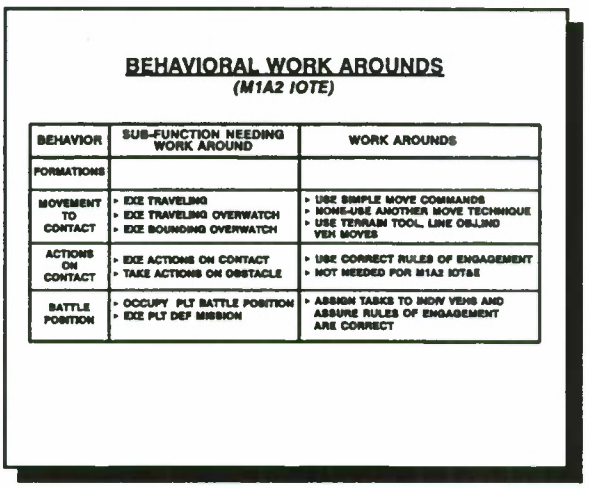


Figure 3. Behavioral Work Arounds

5. A2 ATD Experiment Overview

The purpose of the first experiment was to validate virtual simulation (BDS-D) with live simulation (IOT&E) and to validate constructive simulation (ModSAF and CASTFOREM) with live and virtual simulation. Experiment 1 satisfied the following technical objectives for A2 ATD: the demonstration of DIS as an evaluation tool; VV&A of the M1A2 simulator, ModSAF, and BDS-D; demonstration of analytical tools supporting VV&A and evaluation of simulation outcomes; and demonstration of the M1A2 virtual prototype.

Two battles from the M1A2 IOT&E were replicated, a hasty attack and hasty defense. In each battle a blue platoon of four M1A2s was represented by manned simulators and the remaining ten tanks in the company (two platoons, CO and XO) were implemented by ModSAF entities. Fourteen M1A2s comprised all blue force entities in each battle. In the hasty attack, ModSAF portrayed four T80s and three BMP systems. For the hasty defense, ModSAF portrayed twenty-six T80s and 1 BMP system. Red forces fired anti-armor missiles and sabot rounds while blue forces fired only sabot rounds.

The first experiment was conducted on a DIS local area network at the Mounted Warfare Test Bed at Ft. Knox, Ky. Intercommunication was through ethernet. Components of the network included the four manned simulators, a stealth display, simulation manager, ModSAF Red and Blue commanders' workstations, and data loggers that logged protocol data units traffic during experiment trials and forwarded the logs to the DIS analytical tools.

Before the experiment, detailed evaluation and test plans were prepared and troops were trained. Pilot tests were run to insure that the experiment could be executed and data could be collected and analyzed using DIS analytical tools. The TRADOC accredited stochastic constructive force-on-force combat simulation, CASTFOREM was run prior to Experiment 1 to perform several data checks and comparisons with ModSAF. The employment of CASTFOREM to pre-experiment analysis provided the capability to refine performance and scenario input data. It also provided a means to benchmark the performance of ModSAF throughout the experiment.

The first A2 ATD experiment was credible because the entrance criteria were satisfied. Forty-eight trials were run over a 12-day period. Twenty-four trials were run for each battle. (12 trials with manned simulators and 12 trials with ModSAF only). The platoon locations were randomized to minimize the effects of learning the scenario during the experiment.

6. A2 ATD Experiment Analysis Cycle

The analysis cycle for Experiment 1 is presented in figure 4. The scenario vignette and performance data were fed into both BDS-D and CASTFOREM.

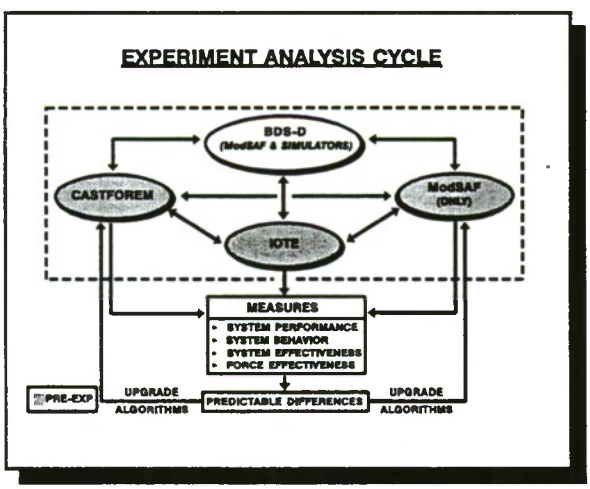


Figure 4. Experiment 1 Analysis Cycle

The BDS-D simulations were run with verified, validated, and accredited ModSAF and simulators (level 2 CIGs) in a level 2 environment with level 2 DIS standards (necessary conditions for conducting experiments). BDS-D simulation runs were made with and without simulators to provide the basis for comparing SAFOR and simulator behaviors and simulation outcomes. For Experiment 1, these outcomes were compared to the outcomes of the M1A2 IOT&E for the replicated battles. ModSAF behavior algorithms will be changed as appropriate for subsequent experiments. BDS-D simulation outcomes were compared with CASTFOREM outcomes to determine the nature of and reasons for differences. CASTFOREM algorithm changes and runs were made to bring the outcomes into better agreement with BDS-D simulation outcomes that were previously VV&A'd to base simulations.

7. References

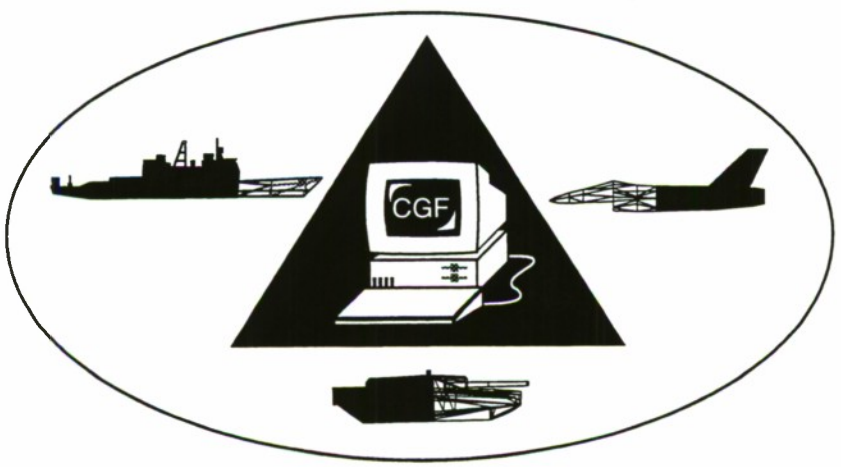
Denney, Carrol (1994),
"ModSAF VV&A Presentation", TRAC-WSMR
Monday, Paul (1995)
"DISVVA Users Manual", Loral ADST
Topper, Phil (1993)
"A Compendium of Close Combat Tactical Trainer Data Structures", US AMSAA

8. Author's Biography

John G. Thomas is an Operations Research Analyst in the Simulation Branch , Combat Integration Division, AMSAA. Mr. Thomas is AMSAA's lead analyst for ModSAF and Close Combat Tactical Trainer (CCTT) SAF VV&A efforts. Mr. Thomas has a Master of Arts degree in Mathematics.

Session 5a: Command & Control Modeling II

Mall, SAIC
Nielsen, University of Michigan
Pratt, NPGS


CGF

Command Entity Cognitive Behaviors for SAF and CGF

Howard Mall mallh@nefarious.saic.com
Kent Bimson bimsonk@nefarious.saic.com
Jenifer McCormack mccormaj@nefarious.saic.com
Dirk Ourston ourstond@nefarious.saic.com

Science Applications International Corporation
3045 Technology Parkway, Orlando, FL 32826-3299

Abstract

This paper discusses the development of a computer-generated *Command Entity* (CE) capable of operating autonomously on a simulated battlefield. The SAF operator's workload would be reduced by extending the reasoning ability of Computer Generated Forces. The construction of a general architecture that allows the integration of heterogeneous AI technologies is described. The system starts with a Knowledge Base (KB) that interconnects both a symbolic (semantic net) and spatial (tactical map) representation of the CE's perception of the battlefield. The KB is maintained and monitored by *Intelligent Agents* that act within their own designated areas of expertise as staff officers to the CE. The interoperation of the CE's components are explained through an example scenario. This work has impact in the areas of mission replanning, command-decision support, and after action review.

1 Introduction

Battlefield commanders analyze Mission, Enemy, Terrain, Troops, and Time (METT-T) in order to do situation assessment. The operation of SAF are divided into *event-driven* and *judgmental* behaviors (Bimson, Marsden, McKenzie, Paz 1994). This paper describes the construction of a computer-generated Command Entity (CE) that can produce the judgmental behaviors needed to increase SAF autonomy. Specifically, the decision-making behaviors of a generic US Army company commander in the mechanized warfare domain is being examined. At this level of command hierarchy the tactical coordination of troops and tanks requires judgment and provides an excellent starting point for this project.

1.1 Judgmental METT-T

METT-T reasoning is the situation assessment process laid out in Army doctrinal literature for their commanders. The basic flow of the commander's decision making process can be seen in Figure 1. Each oval represents a complex set of subprocesses of which METT-T analysis is part. This flowchart represents both pre-exercise planning and the response to discrete events on the battlefield.

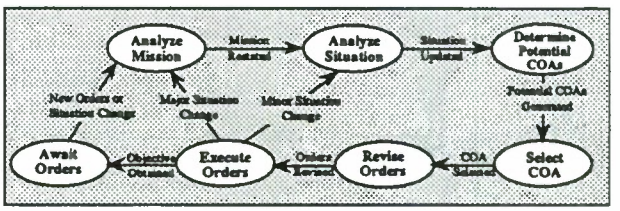


Figure 1. Flowchart of Commander Decision Making Process

These two types of behavior require different cognitive skills. The commanders response to attack by an enemy force is clearly defined and results in an immediate response. There is very little high-level judgment required. The commander considers only his local situation and not the "big picture". Higher-level examination of events requires significantly more sophisticated cognitive skills in order to generate judgmental decision-making.

1.2 Current METT-T Capabilities on CCTT

Current CCTT SAF capabilities fall into one of two categories: (1) executing tasks assigned by the SAF operator and (2) responding to discrete battlefield events. In other words, SAF CE behaviors are invoked in one of three ways within a simulation exercise:

1. Execution of the operations order,
2. SAF operator inputs,
3. Response to situational interrupts.

The SAF operator is in charge of constructing the operations order of the military units and providing new commands to generate realistic human-like behaviors in the simulation. The third operation has been automated with simple "reactive" behaviors in CCTT.

The project began with the event-driven SAF behaviors developed for CCTT and sought to augment them with judgmental METT-T. It was found that in order to develop judgmental capabilities a new architecture that could be integrated into the existing simulation environments would be needed. The CE concept was designed to act as a SAF operator augmentation or

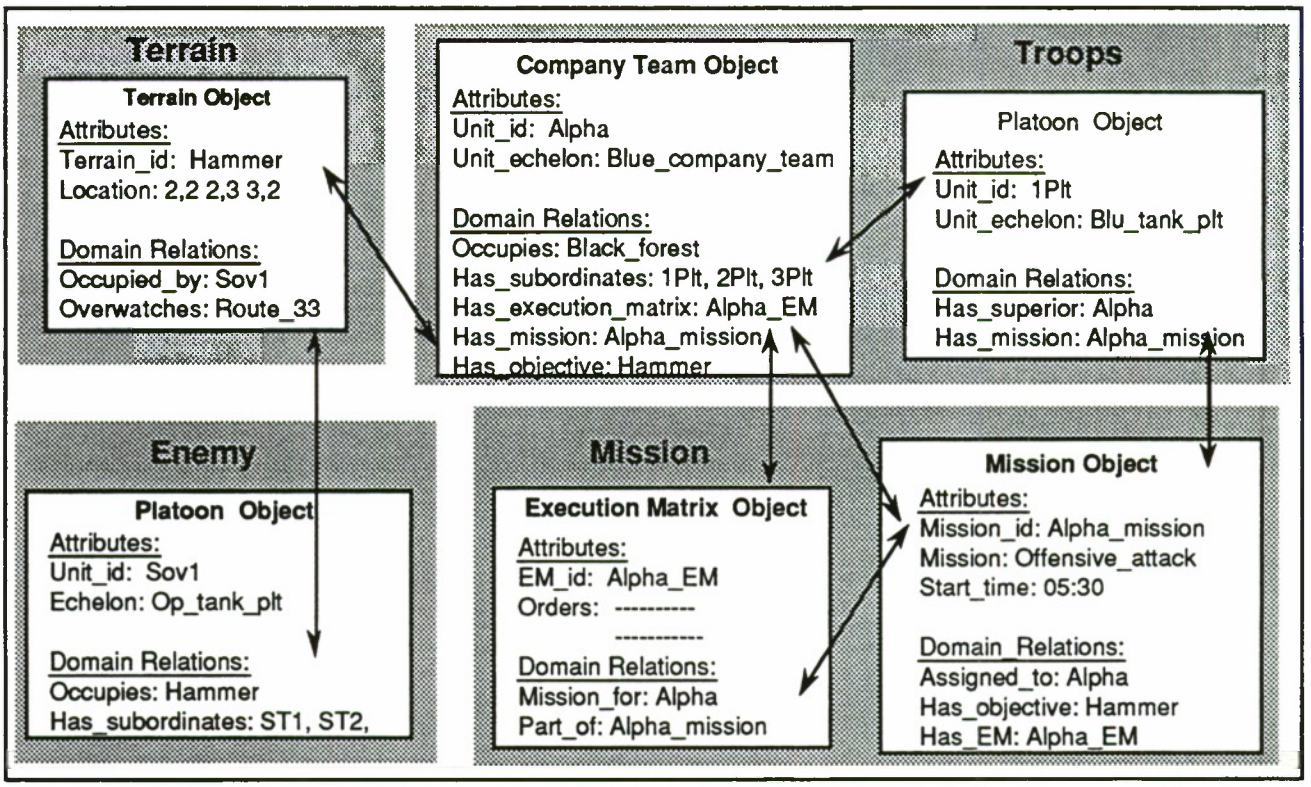


Figure 2. Semantic Network

surrogate. The inherent flexibility and generality of the architecture will allow CE's to be developed with a variety of capabilities and applications to many different simulation systems.

2 Representation

Much as the human brain has two functionally different sides so does the CE's Knowledge Base (KB). It is composed of two structures that represent perceptions of the battlefield: (1) a Semantic Network and (2) a Tactical Map. These structures reconstruct the battlefield based on information the CE has received from reports, orders, and sensor data. The semantic network forms relationships among battlefield objects through the use of symbolic links. The CE's map captures spatial relationships using a two-dimensional array of pointers to objects within the semantic network. This approach effectively aggregates complex information by allowing conduits between the spatial and symbolic relationships among objects.

2.1 Semantic Network

The CE's Semantic Network builds connections between objects in the battlefield environment. Figure 2 is an example of some battlefield objects and how they interrelate. The objects have attributes to store important data. The Terrain object in Fig. 2 has terrain_id and location attributes for identification and spatial orientation, respectively.

Attributes also facilitate domain relationships between objects in the CE's semantic net. For example, a company is composed of three platoons. The Has_subordinates attribute in the Company object lists three instances of the Platoon class to indicate this hierarchical relationship.

The semantic network stores declarative knowledge meaningfully, allowing it to be quickly accessed through the domain relations. The flexible structure of the semantic network helps to easily express the dynamic nature of the battlefield. As relationships between objects change and new objects are introduced the semantic network expands to encompass more information, improving the CE's knowledge of the battlefield.

2.2 Tactical Map

To complete the CE's view of the world, a

representation of its spatial environment is needed. Semantic nets represent conceptual relationships well, but they are inadequate in capturing geographical information.

The Tactical Map coalesces information from terrain assessments and military reports and represents these results in a two-dimensional grid. The cells of this grid contain pointers to objects within the semantic network (see Figure 3). From the Tactical Map, a CE can determine its position on the terrain grid and its position relative to terrain features, enemy positions, friendly forces, objects, etc. Symbolic relationships are traced through the Map's links to the semantic network.

The CE's Map is also a repository of tactical influence factors. The impact of specific objects are numerically determined for Trafficability, Cover and Concealment, and Threat. Each grid square has a 3-tuple to store these factors. The extent of an object's impact is reflected in the grid squares over which it has influence.

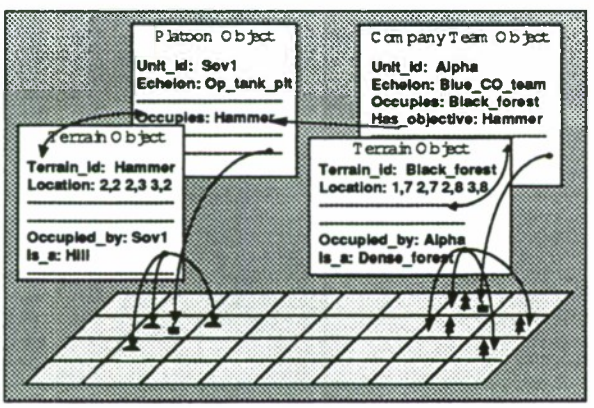


Figure 3. The CE's Tactical Map

Tactical influence factors allow new routes to be quickly generated and selected. The A* algorithm has been used with success (Ourston et al. 1995). Because the three factors are differentiated, they can be assigned different weights for generating routes. Using this technique, routes are generated based on one of the factors or any combination of factors. The factors also act to differentiate candidate routes as a basis for selection among them. For example, speed may be more important than being observed by the enemy; therefore the numerical value for Trafficability carries greater weight that that for Cover and Concealment. The CE would search for and select the most trafficable route versus the one providing more concealment.

3 Reasoning

For tactical situation assessment the US Army has identified five areas that their own battlefield commanders should consider in situation assessment: Mission, Enemy, Terrain, Troops, and Time (METT-T). Commanders plan and replan missions, extrapolate enemy intentions, assess terrain, and manage troops, all within the constraints of time. To produce realistic behaviors within the battlefield domain, the CE must be capable of these cognitive activities. These behaviors require a significant amount of specialized knowledge and analyses which cannot be easily accomplished by any one AI technique. The following architecture provides a facility for applying heterogeneous AI technologies in a cooperative manner.

The CE reasoning is accomplished through Intelligent Agents (IA's) that independently monitor and maintain the common KB. Figure 4 shows how information flows to the CE from the battlefield domain through reports, orders, and sensors (direct observation). The CE communicates with other elements in the battlefield domain through these same channels. As a model of human command judgment this approach approximates the way it is believed actual commanders obtain and process information. Commanders update their complete perception of the battlefield, then they analyze this information from different viewpoints using a variety of cognitive skills. The specialized activities of the IA's blend to generate complex behaviors.

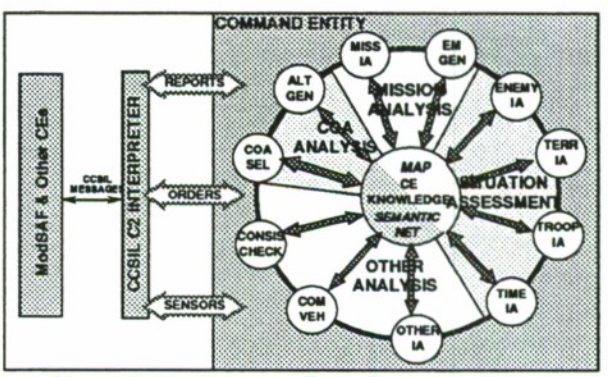


Figure 4. The Command Entity Reasoning Concept

3.1 Intelligent Agents

Specialist Intelligent Agents (IA's) were developed to classify domain knowledge and compartmentalize their reasoning processes. The IA's act independently to build, modify, observe, and analyze the CE's KB. Having the KB common to all of the IA's allows their independent actions to be distributed to other IA's. This communication occurs when one IA modifies a part of the semantic network which another IA monitors. The IA's are also capable of direct dialog in which one IA requests a specialized service from

another.

Figure 5 is the model used to construct an IA. This architecture creates a mechanism for heterogeneous reasoning processes to be executed and integrated. The IA has a defined expertise, a reasoning mechanism, a representation of specialized domain knowledge, a facility for inter-agent communication, and links to the KB.

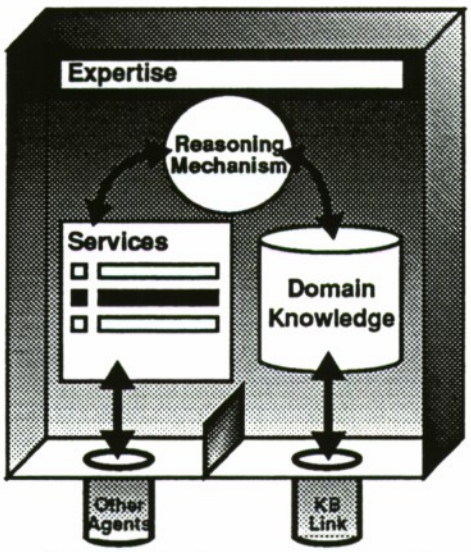


Figure 5. The Intelligent Agent Model

The following discussion will describe the construction of the Mission IA, the Terrain IA, and the Alternative Generator IA as examples of this architecture. These three examples all use different techniques for reasoning about the battlefield situation, employing technology from planning, expert systems, and Case Based Reasoning (CBR). They are currently very simple, yet they cooperatively generate complex behaviors.

3.1.1 The Mission IA

Figure 6 shows the Mission IA based on the intelligent agent model. The Mission IA monitors the battlefield environment to assure that the military units can achieve their goals. A mission is simply a plan. For this reason the Mission IA has a unique viewpoint on the battlefield which differs from that of other IA's. The domain knowledge of the Mission IA expresses its different viewpoint.

The Mission IA's domain knowledge is stored in a conjunctive goal network (Jones, Laird, Tanbe, Rosenbloom 1994 and Wilkins 1988). Each node represents a goal that must be accomplished. The horizontal arcs represent temporal relationships, as in Figure 6 where goal A comes before goal B. The vertical arcs indicate hierarchical dependencies. For instance, goal B will be accomplished when goals C and D are accomplished. This is similar to a PERT chart in project management

The arcs from goal C to the KB "link" in Figure 5 indicate the dependencies of C on attributes within the KB. For example, goal C's feasibility may depend on the size attribute of an enemy object within the CE's semantic network. A link would be established between this attribute and goal C. If the size of the expected enemy is too large, then there is a problem with the mission. The problem is characterized as a force mismatch originating in goal C.

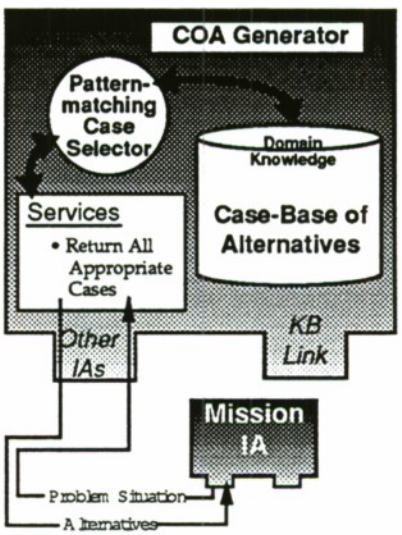


Figure 6. The Mission Intelligent Agent

The reasoning engine of the Mission IA consists of simple rules and procedures that manage the operation of the goal network. The services of the current Mission IA are to compute delays and to return mission critical goals. By developing more sophisticated rules and procedures and expanding the services it provides, the Mission IA can grow without an effect on the other IA's. To increase its capability the Mission IA will use other IA's as a knowledge resource.

3.1.2 The Terrain IA

The Terrain IA is responsible for monitoring and updating information about the terrain elements on the battlefield, such as rivers, hills, forests and cultural features like bridges. Figure 7 shows how the Terrain IA is constructed to carry out its duties. It expresses its domain knowledge with inductive rules that pattern

match on objects within the CE's KB. Its reasoning mechanism is conducted by a forward-chaining inference engine. Using this technique, the Terrain IA provides the services of identifying trafficable areas, obstacles, and the status of dynamic terrain.

The Terrain IA examines reports about the terrain and updates the semantic net, instantiating new terrain objects when necessary. The Terrain IA also updates the CE's map by recalculating any changed influences. For example, the Terrain IA receives a report that a road has been destroyed. The Terrain IA would update its existing road object in the semantic network. The grid squares through which that road passes now have lower trafficability. The Terrain IA calculates the new trafficability score for those grid squares and updates the CE's map accordingly.

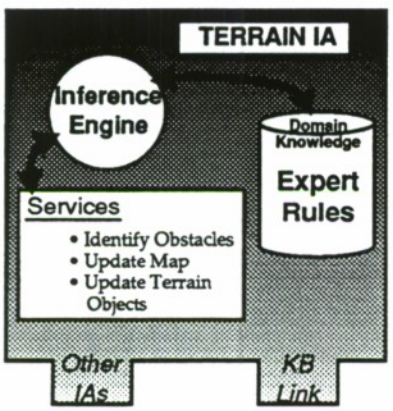


Figure 7. The Alternative Generator

The Terrain IA's analyses are communicated to other IA's through the KB link. Other IA's need only be aware of those changes to terrain that impact their particular specialties.

3.1.3 The Alternative Generator

The Alternative Generator finds appropriate courses of action to given problem situations (Wall 1987). These problem situations are identified by other IA's, as the Mission IA is doing in Figure 8. The domain knowledge of the Alternative generator is expressed using Case Based Reasoning (CBR). The Alternative Generator contains a case library of possible solutions to problems. Its reasoning mechanism matches a given problem situation to one or more appropriate alternatives in its case library.

The problem situation is characterized by its class, its type, and its origin. Alternatives that match the class of the problem situation are general solutions to the problem, while those that match both class and type are more specific. For instance, a problem situation of class *obstacle* would have general solutions, such as *reroute* or *break-through*. A problem situation of type *bridge-out*, however, is an obstacle with more specific solutions such as *find-river-crossing* or *build-temporary-bridge*. Alternatives are further discriminated by the origin of the problem situation. A problem that occurs at a battalion level of command has different options from those at the platoon level. This allows the same case-base to be used by different CE's within a hierarchical command structure. It also allows problems that cannot be solved at lower levels to be referred to higher levels where solutions exist.

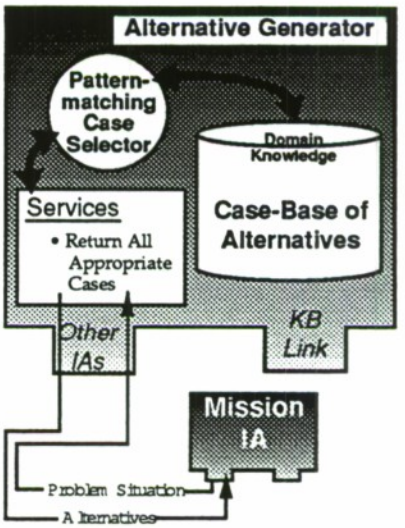


Figure 8. The Alternative Generator

The alternatives produced by this IA identify a number of resources for judging a solutions feasibility. These resources include materiel, personnel, and time. These resources are examined for their availability and if found lacking the solution is judged infeasible. Once feasibility has been determined, then the more specific alternatives have precedence over the general. If no solutions prove feasible, then the problem situation is referred to higher levels of command.

4. Example Scenario

The following example illustrates the interoperation of the CE's components. The CE's objective is to move from point A to point B through battlefield terrain as quickly as possible while avoiding enemy contact. The battlefield contains a forest, a river, and a bridge.

No enemy is currently known to be in the area so a route seeking to maximize Trafficability is generated by the Terrain IA. However, before the mission is begun a

report is received indicating enemy was spotted in the area. The Enemy IA uses the information from the report to instantiate an enemy unit in the Semantic Network and place it on the Tactical Map. The Terrain IA updates the influence factors for Threat. One of the objectives of the mission is to avoid the enemy. The Mission IA, therefore, monitors the Threat influences along the current route. The Threat is now high, so the Mission IA indicates this problem to the Alternative Generator.

The Alternative Generator returns feasible solutions from its case library. The Mission IA determines the delays of any of these alternatives and selects to *reroute*. The Terrain IA generates a new route which has lower Trafficability but also lower Threat. The CE now executes its mission and begins to send orders to move its troops and vehicles.

The route crosses the river at a bridge. As the mission progresses the CE receives a report indicating the bridge has been destroyed. The Terrain IA updates the bridge object. The Mission IA has been monitoring its new route and queries the Terrain IA to define the problem. The Terrain IA recognizes the problem situation of *bridge-out*, which is sent to the Alternative Generator. The Alternative Generator returns possible solutions from its case library. The Mission IA chooses to *find-river-crossing*. One is found, the CE's forces cross to point B, and the Mission IA indicates the successful completion of the mission.

In this scenario, the CE looked at trade-offs between its objectives in order to plan routes through the terrain. It evaluated changes in its situation for their impact on the mission. The CE replanned its mission by analyzing alternatives and making a final decision. In spite of the simple inner workings of each individual IA, their interoperation produces sophisticated behaviors.

5. Conclusion

The architecture presented herein has application in mission reassessment, command-decision support, and After Action Review (AAR). This concept provides a model of CE cognitive behavior. Its use not only generates complex autonomous behaviors to reduce SAF operator workload, but it can be used to explain and understand the cognitive process used in command decision making.

The robust representation of battlefield knowledge found in the CE's KB aggregates a significant amount of data. Through the Mission IA many variables are monitored and evaluated to determine their impact on the mission. This approach determines when the mission becomes infeasible. This is useful not only for automated mission replanning but in supporting the SAF operator or an actual battlefield commander in compiling a vast array of knowledge sources into a simple viewpoint.

Command-decision support can be applied in allowing the CE to operate as an advisor. The CE can generate alternatives suggested by the battlefield situation for an actual commander to consider. The CE could also examine a commander's decisions for viability when stress or fatigue may impair human judgment.

AAR support is currently being examined. The CE is being extended to act as a knowledgeable, automated observer of events on a simulated battlefield. The assessments of the CE can be compared with those of a human participant to provide a view of the human's cognitive process. This could facilitate knowledge acquisition by the CE to improve its capabilities. The architecture developed facilitates easy extension of the CE's capacities.

6. Acknowledgments

The creativity and guidance of Frederic McKenzie and Noemi Paz contributed significantly to this project.

7. References

Kent **Bimson**, Craig **Marsden**, Frederic **McKenzie**, and Noemi **Paz**; "Knowledge-Based Tactical Decision Making in the CCTT SAF Prototype", *Computer Generated Forces and Behavioral Representation Conference Proceedings*, May 1994, pp. 293-306.

Randolph M. **Jones**, John E. **Laird**, Milind **Tambe**, and Paul S. **Rosenbloom**; "Generating Behavior in Response to Interacting Goals", *Computer Generated Forces and Behavioral Representation Conference Proceedings*, May 1994, pp. 317-323.

Dirk **Ourston**, "Maneuver Vector Based Route Planning", *1995 FLAIRS Proceedings.*

Rajendra S. **Wall**, "Case-Based Reasoning for Command and Control", *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, TX, December, 1987.

David Edward **Wilkins**, *Practical Planning: Extending the Classical AI Paradigm*, Morgan Kaufmann Publishers, 1988, pp. 28-37, 121-124.

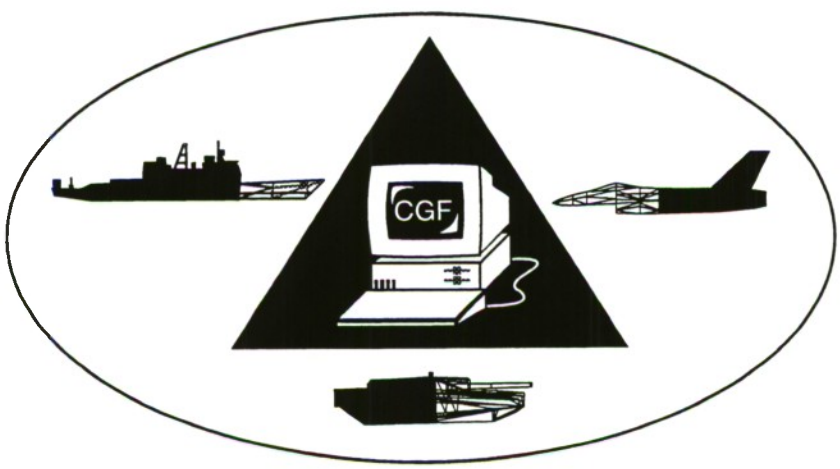
8. Authors' Biographies

Howard Mall is a Master's Candidate in computer science at the University of Central Florida. He currently works on the Command Entity Cognitive Modeling research project at SAIC while completing his advanced degree. He hopes to have completed his Master's thesis by the end of this summer. He has brought an eclectic background with him to SAIC and to UCF. He holds a B.E. in Mechanical Engineering from Vanderbilt University. While there he held a part-time job at the Center for Molecular and Atomic Science at Surfaces doing physics research. He briefly worked a variety of temporary jobs in logistics and manufacturing for Martin-Marietta after college and enrolled in computer science classes at UCF. His research interests are in Artificial Intelligence and Computer Vision, and hopes to continue to contribute in these areas while pursuing a Ph.D.

Dr. Kent D. Bimson is Chief Scientist at Science Applications International Corporation (SAIC) Orlando. Dr. Bimson is in charge of coordinating R&D for SAIC's Orlando office and for business development in support of the group's research efforts. He has been involved in AI research and development activities for 13 years, including work in knowledge-based project management and risk management. Before joining SAIC, Dr. Bimson previously served as a Research Scientist at Lockheed Software Technology Center in Austin, TX, from 1985-1991. He also served as Associate Professor of Computer Science at California State University, Sacramento from 1983-1985, where he taught courses in AI and Natural Language Processing. He holds a Ph.D. in Linguistics from UCLA and a Master of Science in Computer Science (AI) from California State University, Sacramento. Dr. Bimson has published in numerous professional conference proceedings and publications.

Dirk Ourston is currently Principal Investigator for the Command Entity Cognitive Modeling research project at SAIC. Prior to his current assignment he was responsible for the Behaviors software development on the CCTT SAF project (a Semi-Automated Forces simulation being developed for the U.S. Army). Prior to joining SAIC, he worked at BP America, where he was responsible for the artificial intelligence applications for use in the oil industry. He has a Ph.D. in Computer Science from the University of Texas at Austin, with a specialization in machine learning techniques. He has had over 25 years of experience in various types of software applications.

Jenifer McCormack is currently completing her Ph.D. thesis in Industrial Engineering at the University of Central Florida where she also earned her bachelor's and master's degrees. Her dissertation concerns machine learning and student modeling. For a number of years, she worked on a variety of development projects at Harris Corporation. She has instructed courses applying artificial intelligence to industrial engineering. She now contributes her skills to SAIC on the Command Entity Cognitive Modeling research project and at the Intelligent Simulation Laboratory of UCF.


CGF

Intelligent Computer Generated Forces for Command and Control

Paul E. Nielsen
Department of Electrical Engineering and Computer Science
University of Michigan
1101 Beal Ave., Ann Arbor, MI 48109-2110
nielsen@eecs.umich.edu

The clever combatant looks to the effect
of combined energy, and does not require too much from individuals.

Sun Tzu *The Art of War*

1. Abstract

The effectiveness of intelligent computer generated forces is limited by their ability to closely coordinate their actions within the overall battlefield situation. We have developed intelligent command and control agents which monitor large sections of the battlefield and deploy other forces for increased effectiveness. These agents have been demonstrated in the air to air, close air support, and air strike domains.

2. Introduction

Our goal is the development of intelligent forces (IFOR's), computer agents which are functionally indistinguishable from human agents in their ability to interact with the synthetic environment. The Soar/IFOR consortium, involving the University of Michigan, Information Sciences Institute of the University of Southern California, and Carnegie Melon University, is developing IFORs for all military air missions: air to air, air to ground, air supply, anti-armor attack, etc. IFORs must have many capabilities to be successful including: extensive knowledge, real-time reactivity, goal-directed problem solving, and planning. Additionally, they must coordinate their activities with other friendly forces (Laird *et al.*, 1995a).

To fully support very large scale battle field simulations, such as those envisioned for STOW-97, intelligent computer generated forces cannot act independently; but rather, they must coordinate their efforts for increased effect just as humans do. This requires a means and a method for coordination, the ability to convey coordination information, and the ability for large scale situation assessment. In military parlance this is commonly referred to as command, control, communications, and intelligence (C^3I).

This paper discusses our current state of development of intelligent, realistic C^3I agents for simulation in the air domain. These agents have been implemented using ModSAF (Calder *et al.*, 1993) and the Soar/ModSAF interface (Schwamb *et al.*, 1994).

The remainder of this introductory section provides an overview of the C^3I domain and some motivation for this work. Section 3 has a description of the C^3I agents implemented by this project to date. Section 4 discusses the general responsibilities of each agent and goes on to show how our agents demonstrate each of the C^3I functions. Section 5 provides an extended example of the interaction between multiple C^3I agents and a section of planes flying close air support. Section 6 discusses research and open problems. Finally, section 7 provides general discussion and conclusions.

2.1. Domain Overview

Previous work in computer generated forces has either focused on individual agents working in relative isolation or groups of agents which may be treated as a whole (Rao *et al.*, 1994). A notable exception is (Ballas *et al.*, in press). These approaches avoid the problems of C^3I by allowing human guidance, but when the agents number in the tens of thousands, finding enough people to control them is infeasible.

In 1994, the Soar/IFOR project was tasked to provide automated pilots for all air vehicles and missions in support of STOW-97. (See (Laird *et al.*, 1995b) for an overview of the current state of this project.) In order to accomplish this task we needed to extend the scope of the project to include those interactions necessary between pilots and controllers, even if they are not airborne. For example, orange agents are at a severe disadvantage if they cannot rely on ground based radar control (GCI) to track threats outside the limited scope of their own radar.

The most intensive C31 missions we have implemented to date are air to air combat and close air support (CAS). In the air to air domain, the controller may be responsible for maintaining a defensive perimeter

around the carrier battle group, locating potential threats, confirming that an unknown aircraft is a threat, providing timely updates until friendly planes have radar contact, then issuing additional information in response to queries.

While air to air combat has a single (or small number of) controllers, in contrast, the close air support domain demonstrates a wide variety of controllers. In the CAS domain, the attack planes must have detailed integration with multiple agents because of close proximity between targets
and friendly forces. These controllers communicate with the planes (locating targets and deconflicting) as well as amongst themselves (requesting missions and allocating forces.)

2.2. Motivation

The primary motivation for doing this work is to develop realistic C^3I agents. The IFOR C^3I agents should be indistinguishable from human agents performing similar functions. This involves believable interacts with the simulator as well as interactions with other agents and humans at a natural level. By basing IFOR agents on Soar, a theory of cognition (Laird & Rosenbloom, 1994; Laird *et al.*, 1987; Newell, 1987), and modeling not only the externally observable behavior, but plausible thought processes which are necessary to produce realistic behavior, we intend to overcome both dumb, canned responses and implausible, superhuman responses.

The second motivation for doing this work is effectiveness. Without C^3I agents our automated pilots have only limited ability to sense and interact with their environment. Enemy agents can sneak up behind them or fly around them. In addition the automated pilots have only limited ability to change their mission. Without the large scale perspective provided by the controller, they don't even realize that there might be a need to change their mission.

Adding C^3I can increase the level and types of applications for military simulation. As battlefield simulators become more realistic, we want to make them available for more advanced purposes. The major use of air simulators to date is in pilot training. By providing intermediate level controllers, we expect to make simulation usable not only in pilot training, but also in training human controllers to interact with and control these controllers.

Finally, we wish to study human cognition and the ability to model it in Soar. C^3I provides a new domain for this research which suggests more knowledge and exhibits different types of knowledge than that used by aircraft pilots.

3. C^3I Agents

In order to increase realism and promote playability at various levels, we base C^3I on existing techniques currently in use by military organizations and embody them in specialized agents corresponding to military controllers. Thus there is a direct one to one mapping between our agents and humans.

Currently, we have operational versions of the following C^3I agents:

- Air Intercept Controller (AIC) which assigns planes to stations, spots threats, and provides information about enemy planes. The AIC is airborne, situated in a plane with a large radar, such as an E-2C.

- Ground Controlled Intercept (GCI) performs the same sort of mission as an AIC but is ground based and immovable.

- Forward Air Controller (FAC) which locates targets and provides final directions for close air support. Forward air controllers may be either ground based or airborne (FAC(A)).

- Direct Air Support Center (DASC) which assigns aircraft to missions, potentially alters the missions, and hands off attack missions to the FAC. The DASC is ship based, usually on the aircraft carrier.
- Tactical Air Direction Controller (TAD) directs air operations within the Amphibious Operations Area (AOA) prior to the establishment of a DASC. The TAD is also ship based and may be co-located with the DASC.

- Fire Support Coordination Center (FSCC) determines the type of support to utilize (CAS, artillery, naval gunfire). If CAS is determined it generates a Joint Tactical Airstrike Request and coordinates CAS requests with the DASC. The FSCC is ground based within the AOA.

- Tactical Air Command Center (TACC) which provides air traffic control, routing, and deconfliction within the AOA. The TACC is

ground based and usually co-located with the FSCC.

In the following section we explore how agents demonstrate the capabilities necessary for coordinating the behaviors of multiple agents.

4. Responsibilities

In addition to the specific responsibilities of each agent given above there are several general responsibilities associated with C^{3}I agents. These responsibilities are broken out into separate topics, but it must be realized that to work effectively all of these activities must be going on simultaneously.

4.1. Command

C^{3}I agents are responsible for mission initiation as well as tracking and modifying the mission as it develops. Typically the planes will have a prebriefed mission, but often this mission will need to be changed or replaced entirely as the battlefield situation developed. Our command agents can change almost every aspect of a mission including assignment of individual CAP[1] stations, routes, target times, and the final targets.

In order to effectively carry out their command function, C^{3}I agents need to have a command organization. We've observed two different command organizations for C^{3}I agents.

In the air to air domain command is centralized. Either the AIC or the GCI are responsible for all air traffic. These agents provide continuous control and information for many sections of planes. Though there may be multiple controllers acting at the same time they have clearly separated duties, and there is very little interaction.

In contrast, in the CAS domain command is decentralized. As the planes fly through different regions they are directed by multiple controllers, all of which are responsible for the ultimate success of the mission. Though there is still a chain of command, because of limited numbers of radios and limited broadcast range the planes may not be in continuous contact with any single controller.

The controllers in CAS need to coordinate not only the planes, but also themselves. The TACC, DASC, FSCC, and FAC have to form a distributed control network in which mission requests and assignments are propagated through the network.

[1] Combat Air Patrol

4.2. Control

The mission of a controller is to continually assess the situation then allocate, or re-allocate, forces for maximum effect. The combined knowledge of overall mission objectives and threat detection makes controllers uniquely capable of resource allocation. They need to assess the resources available and when future resources might become available, balanced against current and potential threats. They must synchronize their own forces, and their efforts with respect to other controllers. Higher level controllers have to trade off the utility of multiple potential assignments for maximal effectiveness, while low level controllers can only shout louder hoping to increase the priority of their request for resource allocation.

Poorly coordinated attacks can be weak and ineffective. One way C^{3}I agents coordinate is by synchronizing attacks through timing constraints. For example, in the CAS domain, when bombing in tight proximity to friendly troops, timings must be accurate to plus or minus ten seconds to avoid interference with friendly troops.

To accomplish this C^{3}I agents must be capable of real-time reactive planning. Both threats, friendly forces, and messages from other controllers may arrive at any time. The overall battle plan must be incrementally supplemented with new information so that we seize opportunities and knowingly avoid or confront risks.

Soar provides several capabilities which help manage these real-time asynchronous inputs. First, the decision of what to do next is handled through production rules. During each decision cycle all relevant rules are tested and allowed to fire in parallel. Thus the sequence of execution is not fixed.

The real-time is requirement handled by making the speed of operator execution comparable to experimental results in humans (Newell, 1990). Since this can only guarantee soft real-time, our agents will react quickly, but may fail to react quickly enough when faced with overly complex situations, just as people do. Limiting the number of available choices increases the speed of decision making. Soar uses operator subgoaling to provide a context for focusing decisions on information relevant to the current situation. For example, when under attack and bugging out an E-2 might not be

overly concerned with planning the course to its CAP station.

Another way to increase military effectiveness is to decrease the interference from one's own forces. In actual combat (as opposed to simulation) this will have serious morale consequences. The deconfliction duties we've implemented range from air traffic control to route planning to explicitly informing the plane of the location friendly forces.

4.3. Communication

The nature of communication is that commands must be brief, and commands must be clear. C^3I agents must communicate relevant information in a timely and effective manner. Communication can range from simple (e.g., "proceed as briefed" or "negative") to very complex, such as a nine line brief shown in figure 1.

The domain of military communication is well researched, and the military jargon provides a form of communication which is brief yet maximizes the communication of necessary knowledge without undue overhead. We attempt to model C^3I using standardized forms, realistic dialog from actual communications of former pilots, and examples from training manuals whenever possible. We believe that by making communication explicit and based on human communication we can offer an approach to better human interaction and easier evaluation of the results of a simulation.

The approach used by the military, and the approach we've adopted, is to use a shared format for all communication. Complex commands use a standard template to reduce transmission time and ensure all relevant information has been communicated.

To compensate for lost messages and electronic interference we repeat messages until confirmation is forthcoming. The receipt of commands must be confirmed through "roger," or if some action is necessary, by the recipient either "wilco" (will comply) or "negative" (will not comply).

While we have yet to incorporate a general natural language understanding system with TacAirSoar, the commands used are based on the actual English communications used between controllers and pilots in similar situations. This makes it easier to understand the behavior of the IFOR commanders, and allows human communication with the IFOR commanders. In order to communicate with other CGFs we will be adopting CCSIL protocols (Salisbury, 1995).

4.4. Intelligence

The most important responsibility of an air controller is to locate, identify, and track threats. "Timely interception is totally dependent of two factors: early detection and positive identification" (Gunston & Spick, 1983). The need to track the threat arises because enemy agents are eminently uncooperative. Some early failures of our fighter agents acting alone arose because human pilots would feign an attack from one direction, then beam or drop and attack from a different direction. The more powerful radar capabilities of the AIC and GCI makes our agents less vulnerable to these tactics.

Each agent has limited capability. Controllers are limited by weapons,[2] [Though, at least one E-2 pilot considers every friendly plane in the sky his weapon.] maneuverability, and speed when compared with the targets they must defend against. To compensate for this lack of ability they provide greater situational awareness either through proximity (e.g., a FAC) or superior equipment (such as an E-2's radar). They must use this awareness to perform continuous intelligence gathering. Without this information even a veteran pilot may be defeated by a poorly equipped pilot of lesser training.

5. Example scenario

Figures 2 through 8 illustrate some of the interaction between command agents and combat aircraft during a close-air support mission. All of this dialog is taken from a simulation run of a close air support mission.

Our agents include a section of F-14d fighters (lead by Falcon14), a TACC (Icepack), an FSCC (Bronco), a DASC (Mustang) and a FAC (Rattler). Each utterance is preceded by the name of the speaker and the radio frequency used for this communication. The frequencies are color coded to match the encryption scheme used in the communication.

```
Falcon14 (white): Icepack this-is Falcon14
Icepack (white): go-ahead
Falcon14 (white): Falcon14
Falcon14 (white): mission-number 20-059
Falcon14 (white): proceeding-to Elmer
Falcon14 (white): angels 32
Falcon14 (white): time-on-station 1+30
Falcon14 (white): checking-in-as fragged
Icepack (white): roger
```

[2]Though, at least one E-2 pilot considers every friendly plane in the sky his weapon.

```
Icepack (white):  Falcon14
Icepack (white): radar-contact
Icepack (white): cleared-to-enter-aoa
Icepack (white): proceed-as-briefed
Icepack (white): maintain angels 32
Icepack (white): check-in-with Mustang
Icepack (white): on orange
Icepack (white): at Tiger
Falcon14 (white): wilco
```

Figure 1: Mission checks in to AOA

In figure 2 the two planes check into the amphibious operations area (AOA) with Icepack. The exact form of the plane's initial check-in message is specified in the SPINs (SPecial INstructions) and may vary across scenarios, but will convey the essential information 1) who I am, 2) where I am, and 3) what am I doing here.

Icepack recognizes this message and realizes that they are both friendly and supposed to be there. Icepack locates their corresponding blip on radar, gives them permission to enter the AOA, and does not change their mission.

Our TACC is capable of some low level air traffic control. In this case it consists of assigning unique, even altitudes to inbound flights, while outbound flights are expected to maintain odd altitudes.

Finally, Icepack hands off control to the next agent, Mustang, at a pre-briefed radio setting.

```
Rattler (silver): Bronco this-is Rattler
Rattler (silver): immediate-mission
Rattler (silver): target-is tank
Rattler (silver): target-location-is
Rattler (silver): x 127000
Rattler (silver): y 27500
Rattler (silver): target-time ASAP
Rattler (silver): desired-results destroy
Rattler (silver): final-control FAC Rattler
Rattler (silver): on green
Bronco (silver): roger Rattler
```

Figure 2: FAC sends tactical air request to FSCC

In figure 3 Rattler finds itself in the line of unfriendly fire and radios back to the FSCC that it needs support immediately. In addition it provides information sufficient for the FSCC to initiate a Joint Tactical Airstrike Request (JTAR).[3]

[3]We've elected not to include an example of a Joint Tactical Airstrike Request because of its detailed nature. The nine/twelve line brief of figure 1 accounts for less than one sixth of its content by size.

The JTAR includes target type, location, time, and desired results. Note that Rattler has elected to be the forward air controller for the mission and direct the final bombing run. The FSCC supplements this information with coordination and mission data.

In figure 4 Bronco (the FSCC) has determined that close air support is the logical response, and transmits the necessary information from the JTAR to Mustang (the DASC). If this were more realistic, the request would be transmitted in hard copy form rather than over the radio, but we are constrained with the information exchanges allowable through ModSAF.

```
Bronco (orange): Mustang this-is Bronco
Bronco (orange): request-number 28-59
Bronco (orange): immediate-mission
Bronco (orange): target-is tank
Bronco (orange): target-location-is
Bronco (orange): x 127000
Bronco (orange): y 27500
Bronco (orange): target-time ASAP
Bronco (orange): desired-results destroy
Bronco (orange): final-control FAC Rattler
Bronco (orange): on green
Mustang (orange): roger
```

Figure 3: FSCC radios DASC

```
Falcon14 (orange): this-is Falcon14
Mustang (orange): go-ahead
Falcon14 (orange): Falcon14
Falcon14 (orange): mission-number 20-059
Falcon14 (orange): proceeding-to Tiger
Falcon14 (orange): angels 32
Falcon14 (orange): time-on-station 1+30
Falcon14 (orange): checking-in-as fragged
Mustang (orange): Falcon14 this-is Mustang
Mustang (orange): proceed-as-briefed
Mustang (orange): check-in-with Rattler
Mustang (orange): on green at Chevy
Falcon14 (orange): wilco
```

Figure 4: Mission checks in with DASC

In figure 5 the lead plane is approaching a holding point and checks in with Mustang. The plane's check in sequence has the same form as seen in figure 2.

At this stage Mustang alters the mission from its pre-specified course. Even though the planes have a pre-briefed mission, Mustang determines that the new mission is more important and redirects the flight to a new contact point (Chevy) and a new controller (Rattler) for further details.

Figure 6 shows Mustang informing Rattler that help is on the way, who they are, and where to expect them.

Rattler has no radar and will assume a plane approaching from that direction is the expected mission.

In figure 7 the planes finally arrive at the contact point for Rattler and check in according to the format seen in figure 2.

```
Mustang (green): Rattler this-is Mustang
Rattler (green): go-ahead
Mustang (green): expect-cas-mission 20-059
Mustang (green): call-sign Falcon14
Mustang (green): at Chevy
Rattler (green): roger
```

Figure 5: DASC contacts FAC

```
Falcon14 (green): Rattler this-is Falcon14
Rattler (green): go-ahead
Falcon14 (green): Falcon14
Falcon14 (green): mission-number 20-059
Falcon14 (green): 2 F-14d
Falcon14 (green): holding-at Chevy
Falcon14 (green): angels 32
Falcon14 (green): 10 MK82
Falcon14 (green): time-on-station 1+30
Falcon14 (green): no-laser-capability
Rattler (green): roger
Rattler (green): Falcon14
```

Figure 6: Mission check in with FAC

Figure 8 shows Rattler delivering a nine line brief similar to that shown in figure 1. This is an information intensive message which relies on the controller and pilot sharing a common communication model. All and only the necessary values are given sequentially without reference to meaning or line numbers.

What's being expressed here is that the initial point will be Joyce. The heading, in magnetic degrees, from the initial point to the target is 052. The distance from the initial point to the target is 18.6 nautical miles. The target's elevation is 0 above mean sea level. The target's description is a "tank". The target's coordinates are 127000 by 27500 in the X/Y coordinate system of ModSAF. The target will be marked with white phosphor.[4] There are friendlies in the area which are 8000 meters to the south-west. After the attack the plane should egress through Ford. And the attack should commence as soon as possible.

Falcon14 signals that he copies all of that information and agrees to it by repeating the time.

[4]The capability for marking a target does not yet exist.

```
Rattler (green): standing-by
Rattler (green): with-9-line-brief
Falcon14 (green): ready-to-copy
Rattler (green): Joyce
Rattler (green): 052
Rattler (green): 18.6
Rattler (green): 0
Rattler (green): tank
Rattler (green): x 127000 y 27500
Rattler (green): wp
Rattler (green): sw 8000 meters
Rattler (green): Ford
Rattler (green): tot ASAP
Falcon14 (green): ASAP
```

Figure 7: FAC gives 9 line brief

Following this, there are brief exchanges when the planes are spotted, cleared to drop, and for damage assessment.

6. Research Issues in C^{3}I

The development of C^{3}I agents presents several interesting research issues.

From a broader artificial intelligence perspective, C^{3}I presents interesting problems in reactive planning and managing dynamically changing goals in the face of uncertainty. The battle field environment is constantly changing. This requires a fast and efficient architecture to keep up with the speed requirements of the situation as well as a flexible architecture for incremental reasoning and reactive planning.

Most of the planning currently done by our system is reactive planning. In some situations the C^{3}I agents may have some time for decision making and should use this time for more deliberate planning. Recent research explores the possibility of incorporating planning and means-ends analysis mechanisms with our agents (van Lent, 1995; Wray, 1995).

This work is very closely related to distributed artificial intelligence. Since we are basing our work on an existing model which seems to work reasonably well, we can avoid many of the problems of distributed artificial intelligence systems. For example, our agents need not carry out protracted negotiations.

We've demonstrated that a template driven approach to language understanding provides a sufficiently flexible command language for many aspects of communication, but it's not clear how far this approach can be extended. More work needs to be done on natural language understand both for agent flexibility and ease

of use in human computer interaction See (Lehman et al., 1995) for recent work.

Though these agents were prepared to take part in the STOW-E demonstration, during rehearsal they were unable to handle the large number of other agents they saw in the world and crashed. This turned out to be a buffer overflow problem, but suggested several methods for reorganizing the way IFOR agents handle large numbers of inputs. Currently, these IFOR agents will slow down and their performance will degrade as the number of other agents they have to consider increases.

In the immediate future we will address more mundane, but no less critical tasks of tracking fuel states and allocating fuel assets.

7. Discussion

We have described the current state of development of C^3I agents used by Soar/IFOR. We have shown how the agents currently implemented demonstrate the specific aspects of the C^3I domain. Finally, we worked through an example which showed multiple control agents interacting with planes on a close air support mission.

We have demonstrated an ability to cope with incomplete knowledge and incrementally supplement information as it becomes available. This requires continuous situation assessment: commands, threats, and resources may arrive at any time.

We believe that automation must be pushed up the command hierarchy. As the number of simulated agents grows, people will have to supervise larger numbers of agents. We believe that the best way to do this is to emulate the present military command hierarchy. This has the advantage of ease of use (nothing new to learn), effectiveness (it has been proven through centuries of warfare), and ease of understanding.

8. Acknowledgments

This work was done in close cooperation with John E. Laird and Randolph M. Jones.
Thanks to BMH Associates, Inc. for their technical assistance, especially Craig Petersen, Mark Checchio, Tom Brandt, and Bob Richards. This research was supported at the University of Michigan as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Research Laboratory.

9. References

Ballas, J. A. abd McFarlane, D. C., Achille, L. B., Stroup, J. L., Heithecker, C. H., & Kushnier, S. D. in press. *Interfaces for intelligent control systems.* Tech. rept. NRL Technical Report. Washington, D. C: Naval Research Laboratory.

Calder, R., Smith, J., Courtenmanche, A., Mar, J., & Ceranowicz, A. 1993. ModSAF behavior simulation and control. *In: Proceedings of the third conference on computer generated forces and behavioral representation.*

Gunston, B., & Spick, M. 1983. *Modern air combat.* New York: Crescent Books.

Laird, J. E., & Rosenbloom, P. S. 1994. *The evolution of the Soar cognitive architecture.* Tech. rept. Computer Science and Engineering, University of Michigan. To appear in Mind Matters, T. Mitchell Editor, 1995.

Laird, J. E., Newell, A., & Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(3).

Laird, J. E., Jones, R. M., & Nielsen, P. E. 1995a. *Multiagent coordination in distributed interactive battlefield simulations.* Tech. rept. Computer Science and Engineering, University of Michigan.

Laird, John E., Johnson, W. Lewis, Jones, Randolph M., Koss, Frank, Lehman, Jill F., Nielsen, Paul E., Rosenbloom, Paul S., Rubinoff, Robert, Schwamb, Karl, Tambe, Milind, Dyke, Julie Van, van Lent, Michael, & Wray, Robert. 1995b (May). Simulated intelligent forces for air: The Soar/IFOR Project 1995. *In: Proceedings of the fifth conference on computer generated forces and behavioral representation.*

Lehman, J. F., Rubinoff, R., & Van Dyke, J. 1995 (May). Natural language processing for IFORs: Comprehension and generation in the air combat domain. *In: Proceedings of the fifth conference on computer generated forces and behavioral representation.*

Newell, A. 1987. *Unified theories of cognition: 1987 William James lectures.* Available on videocassette from Harvard Psychology Department.

Newell, A. 1990. *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Rao, A., Lucas, A., Selvestrel, M., & Murray, G. 1994. *Agent-oriented architecture for air combat simulation.* Tech. rept. The Australian Artificial Intelligence Institute. Technical Note 42.

Salisbury, M. 1995. Command and Control Simulation Interface Language (ccsil): Status update. *In: Proceedings of the 12th distributed interactive simulation workshop.* Sponsored by STRICOM and the Institute for Simulation and Training (IST) at the University of Central Florida.

Schwamb, K. B., Koss, F. V., & Keirsey, D. 1994 (May). Working with ModSAF: Interfaces for programs and users. *In: Proceedings of the fourth conference on computer generated forces and behavioral representation.*

van Lent, M. 1995. *Planning and learning in a complex domain.* Tech. rept. The University of Michigan, Department of Electrical Engineering and Computer Science.

Wray, R. E. 1995. *A general framework for meansends analysis.* Tech. rept. The University of Michigan, Department of Electrical Engineering and Computer Science.

10. Author's Biography

Paul E. Nielsen is an assistant research scientist working on the Intelligent Forces Project at the Artificial Intelligence Laboratory of the University of Michigan. He received his Ph.D. from the University of Illinois in 1988. Prior to joining the University of Michigan he worked at the GE Corporate Research and Development Center. His research interests include intelligent agent modeling, qualitative physics, machine learning, and time constrained reasoning.

Autonomous Agent Interactions In Modsaf

David R. Pratt, Gary McAndrews, Robert McGhee
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
pratt@cs.nps.navy.mil

1. Abstract

The major problem addressed by this research is the design and implementation of a command and control architecture to add realistic company-level missions to an existing real-time combat-simulation system. The US Army is using the Modular Semi-Autonomous Forces (ModSAF) simulator to conduct research in simulation training (Loral 1993a) (Loral 1993b). While ModSAF is probably the most capable of the Semi-Autonomous Force (SAF) systems in existence, it has the inherit limitation of all such systems. It is primary a reactive system. The missions, or goals, must be set by a higher order controller. This controller can be a computer, Command Forces (CFOR), or a human. However, in the current implementation of ModSAF (1.x), there is no provision for the generation of goals, and hence very little in the way of command and control.

2. Introduction

Since ModSAF is a large system and the range of armored tactics is larger still, we had to focus our efforts into a single company team level task. For this task we chose the "Occupy an Assembly Area" task. At a high level this task can be broken down in subtasks as follows:

The goal of this thesis is to show by proof-of-concept that we can simulate company level tasks utilizing ModSAF's asynchronous augmented finite state machine architecture. The premise is that a finite state machine abstracted to the company commander level can spawn and control existing platoon and vehicle tasks in ModSAF.

The Computer Generated Forces in ModSAF are termed "Semi-Automated" not autonomous. ModSAF behaviors at the vehicle and platoon level exhibit fairly realistic behaviors. However, it is still the responsibility of the operator to provide the realistic interactions between platoons when portraying a higher level unit, like a company or a battalion. One of the design goals was to show that the finite state machine architecture could additionally provide mission planning at the company level, issuing platoon and vehicle instructions to accomplish a mission. Not only will this approach provide more realistic behaviors at the company level, it will reduce the parametric input responsibilities of the operator, allowing him to control a greater number of forces.

The addition of a limited degree of terrain reasoning at the company level, and the ability for units to identify and create their own road routes are features that ModSAF currently does not offer.

3. Task Organization

The task organization for the company assembly area mission includes fourteen M1 tanks, and a utility truck for the First Sergeant. The First Sergeant is given the responsibilities of the quartering party. The purpose of adding the First Sergeant's vehicle was to provide a non-combat type vehicle that performs the reconnaissance and route selections for the company assembly area mission. The M1 tanks of the company continue their current missions while the First Sergeant performs the initial stages of the assembly area mission.

4. Assembly Area Mission Stages

The first step in the design process was identifying a sequence of operations describing the assembly area mission. The sequence of operations used to represent the company assembly area mission are listed in Figure 1.

The first step is identifying the search area for the assembly area. The operator provides this information when the assembly area mission is first assigned. The First Sergeant (1SG) performs the next stages of the mission; planning a route to get to the location, and a terrain reconnaissance of the area. The final stages are performed by the company; movement to the assembly area and occupation of their positions. Each of these actions is discussed in greater detail in the following paragraphs.

Assembly Area (AA) Stages:

- Identification of Assembly Area (AA) Search Location, (operator)
- Planning the Route to the AA Search Location, (1SG)
- Moving to AA Search Location, (1SG)
- Reconnaissance of Search Area, (1SG)
- Designating Unit AA Locations, (1SG)
- Route Planning for Company, (1SG)
- Company Moves to AA, (Company)

Figure 1: Steps for Company Assembly Area

4.1 Identification of Assembly Area Search Location

One of the design goals requires the unit to conduct the mission planning for the assembly area task. The parametric data supplied by the operator for most ModSAF tasks specifies an end goal for a unit. The assembly area mission is somewhat different in its parametric input. Instead of having the operator provide a point location to establish an assembly area, we want a point location that will determine the center of mass of a search area. The unit will determine where to establish the assembly area given the bounding search region. The size of the search area was selected to be a three kilometer by three kilometer square area surrounding the operator's selected center of mass (Figure 2).

4.2 Planning the Route to the Search Location

Many of the ModSAF unit and vehicle tasks -- like move, travel, and assault -- require the operator to input either a point, a line, or some text that provides the parametric input for the task. The unit then performs its task, and often the goal of the task is to move to the point or line designated by the user. There are currently three alternative route inputs; a point location, a line route, and a line route that uses road networks. Regardless of which object the operator utilizes to designate the unit route, it is the operator that supplies the route for the unit.

As part of the design strategy, the assembly area mission attempts to utilize road networks to conduct unit movements. Since it was a design goal to reduce the burden on the operator for providing parametric inputs, like providing the routes for a unit, the unit itself

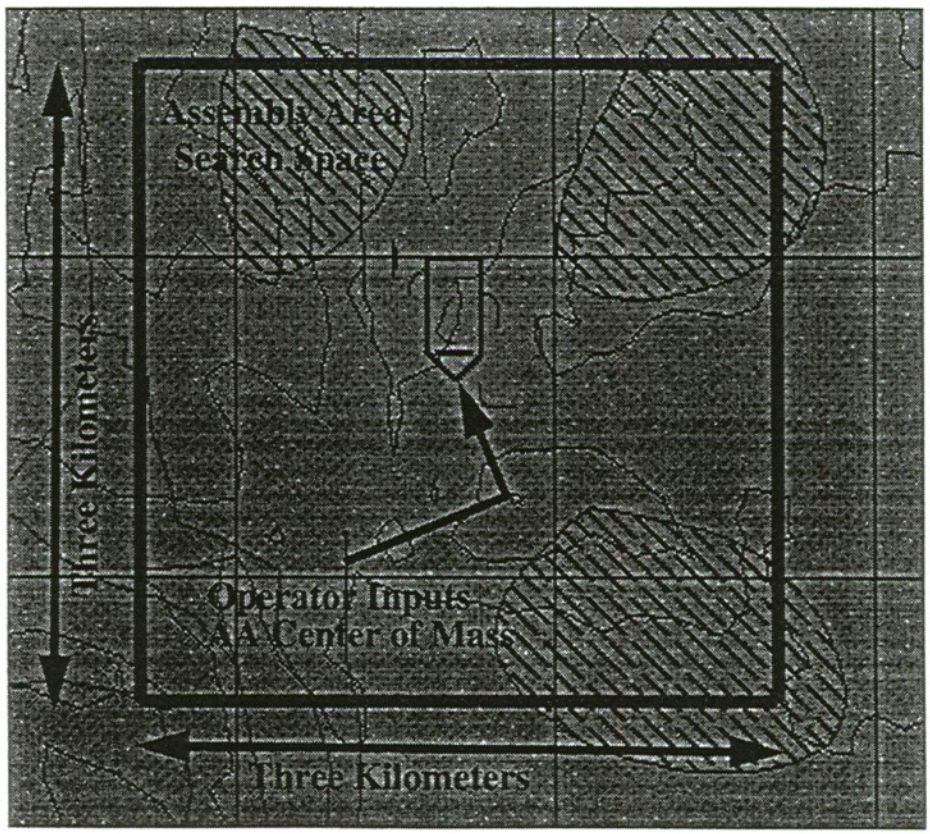


Figure 2: Assembly Area Search Space

should plan its own route. Without modification, however, ModSAF does not provide the functional ability for a unit to determine its own road route. It is a design strategy and goal to provide this capability to ModSAF. The utilization of the road networks is discussed more in Route Planning for the Company.

4.3 Moving to Assembly Area Search Location

Once the First Sergeant identifies a route to the search area, he conducts a ModSAF Vehicle Move Task to get to that location. Where the operator normally provides the parametric inputs for the Vehicle Move task, assembly area routines select the route and pass this information to the Vehicle Move task. The company (minus the 1SG) continues its assigned mission. Once the 1SG arrives at the search area location, he performs a reconnaissance of the area.

4.4 Reconnaissance of the Search Area

After the 1SG arrives in the search area, he begins a reconnaissance. He is looking for areas that provide sufficient space and ideally, both cover and concealment. For the purposes of this mission, we narrowed the search criteria to include only tree canopies large enough to support a company assembly area. The result of this reconnaissance is either the identification of a tree canopy large enough to support a company sized assembly area within the confines of the search area box, or the center of mass location provided by the user.

4.5 Designating Unit Assembly Area Locations

In a company assembly area, each platoon is given a "piece of ground" to occupy within the boundaries of the assembly area. The platoon positions within the assembly area must provide 360 degree security for the company. The amount of space allocated to each platoon depends on the overall size of the assembly area. We can vary the size of our assembly area based on terrain constraints. The minimum radius for our assembly area was chosen to be thirty meters. The maximum, and default assembly area radius, is 250 meters. By distributing the three platoons (twelve company vehicles) in thirty degree increments around a 360 degree assembly area, the platoon locations provide all around security for the company. The platoon occupation positions are sized in accordance with the minimum radius of the selected assembly area. An example diagram of an assembly area is shown below in Figure 3. The headquarters vehicles -- 7, the First Sergeant, 66, the Company Commander and 65, the Executive Officer -- orient their positions facing the enemy direction. The platoon positions on the perimeter of the assembly area are not changed with respect to the enemy's direction.

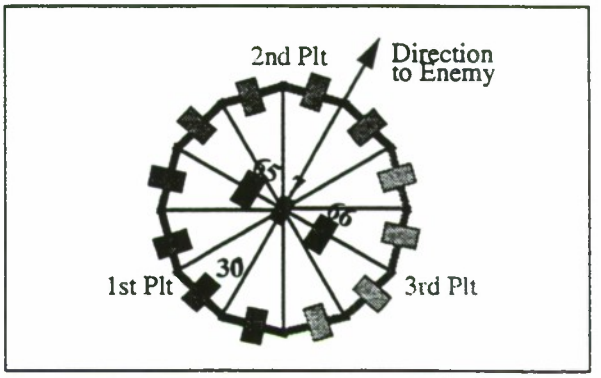


Figure 3: Occupation Positions for Assembly Area

4.6 Route Planning for the Company

After the 1SG completes his reconnaissance, he has either found a suitable tree canopy location for the company, or establishes the assembly area at the center of mass location provided by the operator. He then identifies and selects a road route for the company. The route will include a start point (SP), a release point (RP), and the route itself, the three basic control measures utilized for a military move operation.

When the operator desires to designate a route for a unit, he uses ModSAF's line editor. A line editor option is to generate a road route from the operator's input. The operator selects a start point and end point for the route. ModSAF utilizes an A-star search to identify a road route that connects the start point and end point, if one exists. The result of this function is a road route that gets stored as an object in the PO Database, or a system error message stating a road route could not be found. It was proposed that instead of having the operator use the line editor to specify a unit's route, that the unit have access to the road route building functions and build their own road routes.

4.7 Company Movement to the Assembly Area

The company road march is conducted as individual platoon road marches. The platoon closest to the assembly area moves first, and reports when it has arrived at the start point (SP). The commander, the second to move, then begins his movement. The commander in turn reports the SP and the next closest platoon begins its move. The company road march continues as determined by the company order of march. By breaking the company movement into platoon and vehicle moves, the assembly area mission can

utilize the existing ModSAF Unit Travel Task -- a platoon level task.

4.8 Occupation of the Assembly Area

The assembly area mission uses the ModSAF "Occupy Position" task. Each platoon and headquarters tank is given a position to occupy within the assembly area. The assembly area support functions pass the required parameters to the occupy position task, which includes a line object representing the position to occupy, and three target reference points (TRPs), a left TRP, a right TRP, and an Engagement Area TRP. These TRPs are used to designate sectors of responsibility for each platoon.

5. Modsaf Vehicle, Unit, And Reactionary Tasks

In addition to the ModSAF Vehicle Move Task used by the First Sergeant, and the Unit Travel and Occupy Position Tasks used by the platoons, several vehicle level tasks including collision avoidance, path planning, sensor, turret, gun control, and the unit reactionary task, "Actions on Contact", are used by all of the vehicles when performing the assembly area mission.

6. Finite State Machine Architecture

The development of a finite state machine that represents the company mission assembly area was derived from Figure 1, Assembly Area Stages. The resulting finite state machine used for this mission is shown in Figure 4. The reverse path from "Moving_To Recon"

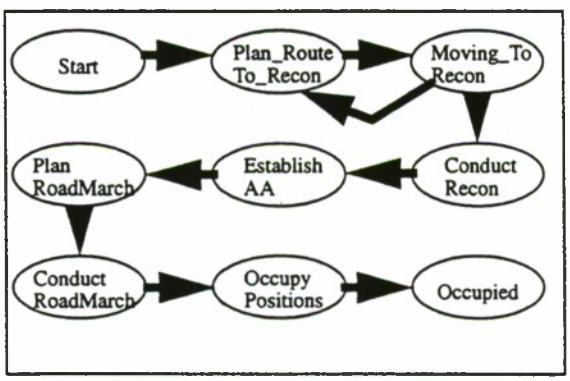


Figure 4: Assembly Area Finite State Machine

back to "Plan_Route_To_Recon" was a design strategy that allows the First Sergeant to plan multiple routes during his reconnaissance mission. He first plans a route to get to the assembly search area. He then plans a route from that location to the closest tree canopy that fulfills the requirements for the assembly area location.

7. Assembly Area Library Module

The result of the design for the company assembly area mission will be an independent ModSAF library module, "Uassembly". The design plan was to currently limit its execution to the specific task organization of fourteen M1 tanks and a First Sergeant's vehicle. The assembly area library module uses the same architecture and design as the other unit level tasks currently implemented in ModSAF.

8. Communication Between Autonomous Agents

The assembly area mission attempts to capture the battlefield communication between the platoons, the Company Commander and the First Sergeant. The ability to display the intercommunication between units exists in ModSAF and was used within the assembly area mission.

9. Task Organization

ModSAF offers an M1 tank company as one of its unit types. The assembly area mission requires adding a new unit type, "M1 Company w/1SG", which includes fourteen M1 tanks and a First Sergeant's truck. Adding this new unit organization requires modifying "echelon.rdr" in the ModSAF Library Module "Libechelondb". The "echelon.rdr" file is a ModSAF "reader file". A reader file is a text based file that allows easy modification by the developer when changing the parameters for ModSAF. The Libechelondb library provides standard military echelon unit organizations, from section to battalion and higher. A unit organization is developed by adding vehicle leaf nodes or other unit tree nodes to the organization. A tree node is another unit organization previously defined in "echelon.rdr" and is recursively expanded when the unit is created. The echelondb format for an M1 Armor Platoon is:

```
(unit_US_M1_Platoon (
    (leaf vehicle_US_M1 "??1")
    (leaf vehicle_US_M1 "??2")
    (leaf vehicle_US_M1 "??3")
    (leaf vehicle_US_M1 "??4"))).
```

Using the echelondb format, creating a new unit entry for "M1 Company w/1SG" is:

```
(unit_US_M1_Company (
    (leaf vehicle_US_HUMMV "?7")
```

```
(leaf vehicle_US_M1 "?66")
(leaf vehicle_US_M1 "?65")
(tree unit_US_M1_Platoon "?1 ")
(tree unit_US_M1_Platoon "?2 ")
(tree unit_US_M1_Platoon "?3 ")))).
```

The characteristics for a specific vehicle type are contained in parametric reader files. These files are located in the "entities" subdirectory. An example model parameter file for the US M1 Abrams Tank is contained in Appendix A. Since these files are reader files they can be tailored by the operator for a specific application. The model parameter file for the US HUMMWV, the First Sergeant's vehicle, was not included in version 1.0 of ModSAF, and was therefore added to the existing set of vehicle model parameter files.

10. Development Of The Finite State Machine Code

The first step in creating the Unit Assembly Area Mission was to create the library module directory for the task. This library was named "libuassembly" and was included in "/modsaf/common/libsrc". The finite state machine file for the unit assembly area mission was named "uassembly_task.fsm" and the task named "uassembly". After designing the finite state diagram, we considered which existing ModSAF tasks the assembly area finite state machine would utilize to conduct its mission. These ModSAF tasks become subtasks for the uassembly task.

10.1 Vehicle Tasks

The First Sergeant moves on his own to the assembly area search site. This being a single vehicle move, the existing ModSAF task Vehicle Move was selected for inclusion as a subtask. Additionally, the reactive tasks for "Actions on Contact" were included as a subtask in the "taskframes.rdr" file for the uassembly task. By including the reactive task in the "taskframes.rdr" file, all entities within the uassembly task have these reactive tasks running concurrently with the assembly area tasks.

10.2 Platoon Level Tasks

Two of the existing ModSAF unit tasks were selected as subtasks for the assembly area mission. The first was the Unit Travel task. Each platoon, and both of the headquarters tanks, use a Unit Travel task to perform the company roadmarch phase of the mission. When the units arrive at the assembly area location, they occupy their respective positions within the assembly area. The ModSAF task "Occupy Position" performs this task sequence. Chapter V describes the occupation task for the assembly area mission.

10.3 ModSAF Finite State Machine Protocol Language

The Assembly Area finite state machine is written in accordance with the specifications described in the LibTask Programmer's Guide, 1993. The Programmer's Guide sets forth a finite state machine protocol language which allows the use of the ModSAF Asynchronous Augmented Finite State Machine (AAFSM) Code Generator. The code generator converts a finite state machine source file into C code. Thus, a simple finite state machine protocol language is utilized to describe the structure of the task. The supporting routines, written in C, are added by the developer to support the behaviors of his task.

11. Finite State Machine Support Routines

The finite state machine support routines for a ModSAF task are created by the developer to perform the desired actions while in a particular state. In the following paragraphs we discuss the support routines needed to perform the actions for each state of the finite state machine.

11.1 Plan Route to Recon

The assembly area mission begins in the state ***"Plan_Route_To_Recon"***. The purpose of this state is to:

- Establish the search space for the assembly area,
- Create a graphic entry in the unit's overlay,
- Establish a route for the First Sergeant to get to this location,
- Create the route graphic for the First Sergeant's overlay, and
- Spawn a ModSAF Vehicle Move Task for the First Sergeant.

The support functions for this state include:

- "compute_recon_route",
- "create_new_ route_to_objective_com", and
- "create_aa_bound_box".

We transition to the next state, ***"Moving_To_Recon"*** after spawning the First Sergeant's Vehicle Move (VMOVE) task.

One of the fundamentals of developing a ModSAF task is the ability to relate graphic control measures depicted on the terrain map display with a particular unit or entity. ModSAF maintains a unit overlay for each entity it simulates. When the operator selects a unit, the overlay for that unit is displayed on the terrain map. We wish to add a bounding box graphic in the company overlay for the assembly area mission. When the operator selects the assembly area mission, he is asked to designate a center of mass location for the search area. This location is stored in a private variable "private->objective_com" associated with the unit. The center of mass (COM) for the search space is used by the function "create_aa_bound_box" to create a graphical box entry in the company overlay.

Once the data for the line object has been entered, we must save the line object into the PO Database -- the database ModSAF utilizes to store the necessary information about the entities it is simulating. We want to check to find if their already exists an assembly area bounding box. If there exists a box, we update its position. If it does not exist, we create a new object and store it in the PO Database. We save this line object as a state variable for the company as "state->assy_area". That way if we change the location for the assembly area, we have a state variable we can lookup and modify.

11.2Moving to the Reconnaissance Site

The purpose of the state ***"Moving_to_Recon"*** is to monitor the progress of the Vehicle Move task spawned by ***"Plan_Route_To_Recon"***. The support function for this state includes "send_arrival_report". We transition to the next state, ***"Conduct_Recon"*** after the First Sergeant's Vehicle Move (VMOVE) task transitions to an "arrived" state.

The Vehicle Move task is an existing ModSAF unit task and includes several states of its own as shown in Figure 6. The previous state, ***"Plan_Route_To_Recon"*** spawns the Vehicle Move task and then switches the state of the assembly area finite state machine to ***"Moving_To_Recon"***. When the First Sergeant arrives at the search area, the Vehicle Move task transitions to the "arrived" state. ***"Moving_To_Recon"*** sends an arrival report to the commander, and transitions to the next state,

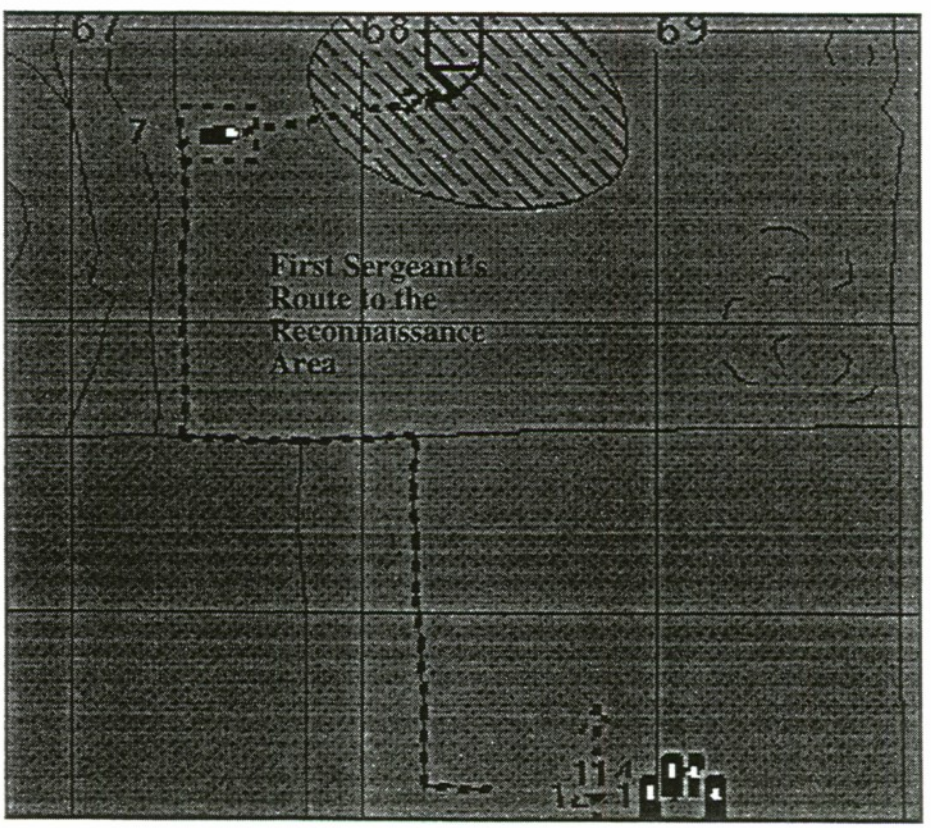


Figure 5: Moving to the Reconnaissance Area

"Conduct_Recon". Figure 5shows the company assembly mission's *"Moving_To_Recon"* state

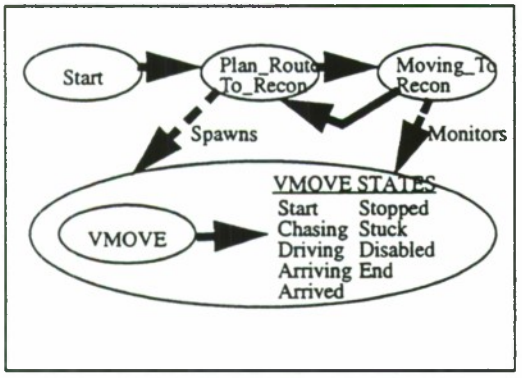


Figure 5: State Plan Route to Recon

11.3Conducting the Reconnaissance

The purpose of the state *"Conduct_Recon"* is to identify the best location to establish an assembly area within the confines of the assembly area bounding box. The support function for this state is "search_for_tree_canopies". If a tree canopy is found within the bounding box that is large enough to support the assembly area, we transition back to *"Moving_To_Recon"* and move to the tree canopy. Otherwise, we did not find a tree canopy and we establish the assembly area location at the center of mass of the search area, and transition to the next state *"Establish_AA"*.

One of the design goals of the assembly area mission was to add a degree of company level mission planning. There are currently only a few terrain analysis functions available in ModSAF. We incorporated a search of the QUAD Tree database for tree canopy locations and a very simplistic algorithm that estimates the center of mass location for the tree canopy. The goal here was not to perfect the terrain analysis algorithm, but to use it as the means to demonstrate the ability of a unit to conduct some level of autonomous mission planning.

Tree canopies are features contained in the Quad Tree Database of the ModSAF "terrain" library. A tree canopy is composed of a set of points that establishes the perimeter of the canopy. Very simply, we average these point locations to estimate a center of mass location for the canopy. This does not work for concave canopies. When a canopy extends beyond the boundaries of our assembly area bounding box, we do not consider those points in the center of mass calculation. The result is a very unsophisticated algorithm that estimates the center of mass of tree canopies, providing a limited terrain analysis ability for tree canopies that currently does not exist in ModSAF.

11.4Establishing the Assembly Area

The purpose of the state *"Establish_AA"* is to create the graphic entries for all of the unit's "Occupy Position" tasks. The support function for this state is "build_assembly_area". Once the positions are created, we transition to the next state, *"Plan_Roadmarch"*.

The function "build_assembly_area" establishes the occupation positions and target reference points for each of the platoons and headquarters tanks, and stores these graphic entries in the respective unit overlay. These graphics are required for the ModSAF Occupy Position tasks.

11.5Planning the Roadmarch to the Assembly Area

The purpose of the state *"Plan_Roadmarch"* is to establish a route for the company to get to the AA, and create the route graphic for the company overlay. The support function for this state is "build_roadmarch". After building the route for the company, we transition to the next state, *"Conduct_Roadmarch"*.

First we discuss adding the functionality to ModSAF that will allow a unit to develop its own road route. Then we discuss how we utilize this new functionality to establish the road routes for the First Sergeant and for the company.

First, we needed to determine what ModSAF used to allow the operator to implement a road route. We analyzed the C code of "edt_line.c" in the "Libeditor" directory of the ModSAF source library. Building a road route consists of reading in the operator's mouse location as he designates the start point and updating the road route as the mouse is moved towards the end point. The parametric inputs are the X and Y locations of the mouse as the operator is making the route, and are passed to a function "rt_allocate_road_route_from_networks" contained in "rt_roads.c" of the "Libroute" directory. This function searches the Quad Tree Database and selects the closest road segment for the starting road point, and using an A-star search, attempts to make a road route that connects the start point to the current position of the mouse cursor. The "rt_allocate_road_route_from_networks" function takes as input arguments the Quad Tree Database being used, the segment number of the closest road seg-

ment, the start location, the segment number of the ending road segment, and the ending location. It then attempts to create a road route that starts at the start point, gets on the road at the near segment, travels on the road to the far segment, and moves to the end point. The function returns a "ROUTE_LIST" which describes the road route in terms of the road segments, the individual points of the road segments used for the route, and an ordering of these points.

The parameters that we did not currently have to make a direct call to "rt_allocate_road_route_from_networks" were the segment numbers of the near and far road segments. We found a function "find_nearest_segment" in "road_routes.c" of the "Libquad" directory. This function, however, was an internal private function. We needed the ability to determine the segment numbers of the nearest road to a given location which this function provides. We therefore added "find_nearest_segment" to the header file for "libquad.h" making it a publicly accessible function. With these modifications, we could utilize "rt_allocate_road_route_from_networks" directly. By providing the unit's current location as the start point, the desired ending location for the end point, and the return road segment numbers from calls to "find_nearest_segment" for both of these points, ModSAF will build a road route for our unit.

In the design strategy, we decided that the road route should include a start point (SP), a release point (RP), and the road route itself. The RP was selected to be the last road point of the road route. The route object was the "ROUTE LIST" returned by the call to "rt_allocate_road_route_from_networks". The start point had to be "massaged" to prevent a problem encountered when using ModSAF's Unit Travel task. When a unit is assigned a route and performs the Unit Travel task, it first determines the "optimal" starting point to get on to the route. This "optimization" could lead a unit to skip the start point, which is not allowed in a standard military road march. Travelling to the start point when conducting a military road march is not an option.

11.6Moving to Assembly Area

The purpose of the state ***"Conduct_Roadmarch"*** is to conduct the company's roadmarch to the assembly area and act as an abstracted command finite state machine. The supporting functions for this state include: "check_for_SP", "send_sp_report", and "send_rp_report". We transition to the next state, ***"Occupying_Positions"*** when all of the units have completed their Unit Travel tasks.

The assembly area state ***"Conduct_Roadmarch"*** not only controls the movement of the company to the assembly area, but also monitors and assigns the appropriate task to each platoon depending on their situations. It is this part of the assembly area finite state ma-

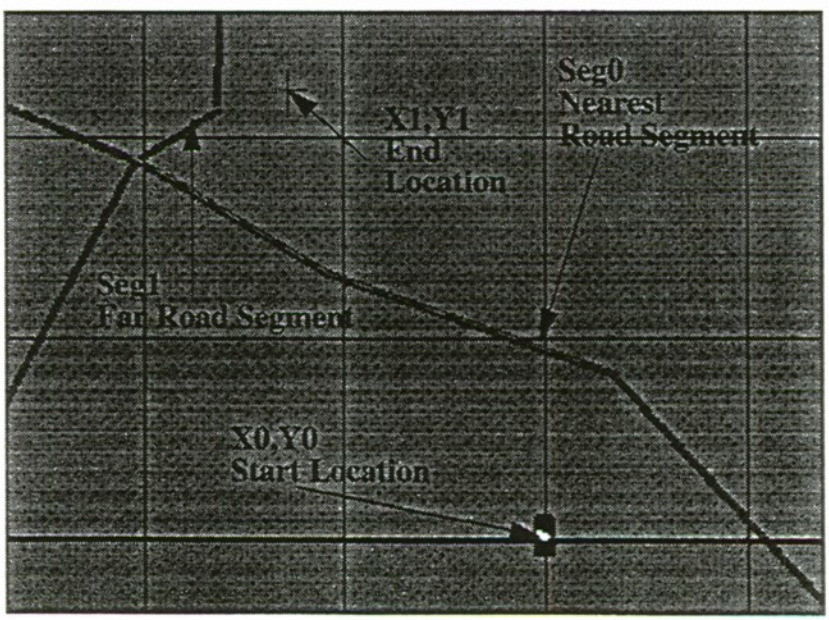


Figure 1: Allocating a Route From Roads

chine that makes it unique from most ModSAF unit tasks. In a typical ModSAF unit task, we only change states when the unit performing the task meets the transition requirements of the task. For instance, if a unit is given a mission to move to a point and then occupy a position, that unit must complete its movement before it can transition into occupying the position. If we applied this rigidity to the assembly area mission, the company road march must be completed by the entire company before the first platoon begins its occupation of the assembly area. We could end up with a company of tanks in a column formation sitting on the road waiting for the last platoon to finish its roadmarch. In an actual roadmarch, the platoons do not stop at the release point, but begin occupation of the assembly area as soon as they arrive at the release point. This is in compliance with (ARMY FM 71-1 1988) which states: "Move vehicles from Release Point (RP) into assembly area without stopping." Therefore the ***"Conduct_Roadmarch"*** state of the assembly area finite state machine acts as a sequencer for the subordinate unit's tasks, and abstracts the command and control of the company -- like the control provided by the Company Commander. It is this command and control element that we wished to capture in the assembly area mission.

The roadmarch is actually five separate Unit Travel tasks -- one for each platoon, and one for each headquarters tank. The order of march for the company is determined according to the proximity of the platoons to the assembly area location. The closest platoon becomes the lead platoon. The Commander (66 tank) follows the lead platoon. The next closest platoon is next in the order of march and is followed by the Executive Officer (65 tank). The platoon farthest from the assembly area is last in the order of march.

The movement of the individual units is monitored and controlled by the ***"Conduct_Roadmarch"*** state of the assembly area finite state machine. When the lead platoon arrives at the SP location, it sends a report to the commander ("send_sp_report") and then the Commander begins his movement. Likewise, when the Commander reaches the SP, he communicates this to the company and the next closest platoon begins its movement. In this way, we maintain adequate spacing between the units and control their movement along the company route.

The command level abstraction of the assembly area mission is most prominently displayed in the transition between the company beginning its roadmarch and then occupying the assembly area. The purpose is to allow the platoons to independently change states while the company commander monitors their states. This was a design implementation that we felt provided an abstracted command and control finite state machine. In fact, there typically is a time during the assembly area mission when the lead platoon is occupying the assembly area task, the next platoon is conducting its roadmarch, and the last platoon is waiting to begin its roadmarch.

11.7 Occupying the Assembly Area

The purpose of the state ***"Occupying_Positions"*** is to monitor the status of the unit Occupy Position tasks spawned in the previous state, ***"Conduct_Roadmarch"***. The supporting function for this state is "clean_march_points". We transition to the next state, ***"Occupied"*** when all of the units have finished their Occupy Position tasks. The "clean_march_points" function is a utility that removes the graphic entries used to conduct the company roadmarch to the assembly area.

11.8 Unit Occupied Assembly Area

The purpose of the state ***"Occupied"*** is to maintain the assembly area location until the operator terminates the mission. This state required no supporting functions and transitions to the ***"End State"*** when the operator terminates the mission or selects a different mission for the unit.

11.9 Ending the Assembly Area Mission

The purpose of the state ***"End"*** is to remove from the Persistent Object Database the graphic entries introduced for the occupy position tasks. The support functions for this state are "clean_occupy_pts" and "delete_graphic_com". The assembly area finite state machine exits after completing this state.

12. Tasks And Task Frame Management

A company level task cannot be generalized to being simply a collection of three identical platoon level tasks running concurrently. As discussed earlier, in the assembly area mission, the company first conducts a road march to the assembly area location and then occupies the location. We do not, however, want the entire company to complete the roadmarch before beginning the occupation of the assembly area. The platoons are conducting different individual missions that encompass the overall company level mission. We need a mechanism to decide the particular unit task an individual platoon should execute. The problem with attempting to generalize a company level mission from

a like platoon level mission can best be explained using a company attack. A platoon on its own conducting an attack may maneuver and fire towards the enemy. A company attack, however, is not simply three platoons conducting concurrent attacks on the same enemy location. The company commander must analyze the enemy situation as soon as the lead platoon makes contact. He may then decide to establish an attack by fire position for the lead platoon and maneuver the remaining two platoons to capitalize on the enemy's weak side. It is proposed that the command finite state machine as used by the assembly area mission could interpret the changing environment (identify an enemy weakness) and initiate the necessary platoon level actions to capitalize on this weakness.

13. Assessment Of Assembly Area Mission

13.1Route Planning

The ability for a unit to determine its own road route given a starting location and and ending location was added to ModSAF. We attempted to portray a standard military road march which includes some basic control measures. This included (as a minimum) a start point, a route, and a release point. We discussed having to work around some of the "features" that ModSAF provides in its Unit Travel task, like the optimization of the entrance point to a route that conflicts with having to cross the start point before using the route. Future developments in ModSAF may allow the designation of an absolute starting point for the route, which would more closely replicate a standard military road movement.

13.2Abstracted Command Finite State Machine Architecture

We used a finite state machine architecture that abstracted the command and control of a company level mission by controlling the independent platoon tasks which make up this mission. Our implementation of the finite state machine architecture is different from most existing ModSAF tasks. Adding real-time mission planning to a company level mission more closely resembles the Company Commander's real-time decision making process occuring on the battlefield. With more research, the use of a command-level finite state machine may be shown to be instrumental in controlling company level tasks as a collection of independent platoon level tasks. The trade off of real-time mission planning is the potential time delays introduced into a real time simulation system. However, without a greater degree of autonomy, the goal of alleviating the operator from providing the necessary realism between platoons and companies will prevent him from replicating a force much larger than a battalion.

14. Conclusions

The primary purpose of this research was to establish a proof-of-concept that higher level tasks, specifically company level tasks and missions, could be developed and incorporated into ModSAF. The result is a prototype company level mission -- Occupy an Assembly Area -- using the finite state machine architecture of ModSAF 1.0. This mission provides realistic timing constraints, communication amongst the autonomous agents, and an abstracted commanding finite state machine that provides the building blocks for the more complex company-level missions Attack and Defend. The limited complexity of the Assembly Area mission permitted rapid development and testing while utilizing the finite state machine architecture. The individual behaviors of the autonomous agents could be individually analyzed and selectively modified.

In our developed Company Assembly Area mission, the First Sergeant, operating independently of the company, conducted a reconnaissance type mission with specific parameters -- identifying a suitable assembly area location. The Company Commander, communicating with the First Sergeant, developed a company road march plan, integrating the individual platoons' road marches into a company level road march. The ability to control multiple platoons performing different unit level tasks is demonstrated in the Assembly Area state ***"Conduct_Roadmarch"***. The ability to control platoon level tasks at an abstracted company commander level utilizing ModSAF 1.0's current finite state machine architecture is both possible and promising.

The analysis of the existing ModSAF architecture, the development of a new company level mission, and the testing and evaluation of this mission leads us to draw the following conclusions:

- ModSAF entities require additional terrain reasoning algorithms.
- ModSAF entities should perform some degree of mission planning.
- Company level missions should be designed as a collection of independent platoon level tasks.
- The current AAFSM architecture of ModSAF can be utilized to develop realistic company-level missions.
- An abstracted command-level finite state

machine controlling the platoon level finite state machines is one approach to higher-level command and control.

15. Acknowledgments

The work described in this paper was funded by STRICOM. The work would not have been possible with out the help of Mr. Joshua Smith and Dr. Andrew Ceranowicz, both of LORAL-ADS.

16. References

United States Army, Field Manual 71-1, The Tank and Mechanized Infantry Battalion Task Force, Headquarters, Department of the Army, October 1988.

Loral ADS, "ModSAF Software Architecture Design and Overview Document," 1993.

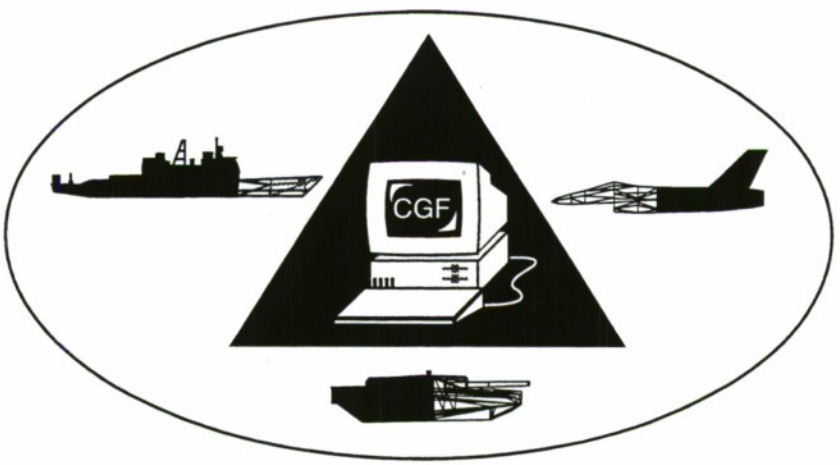
Loral ADS, "A Modular Solution for Semi-Automated Forces -- ModSAF, An Overview," Loral ADS Briefing Slides, 1993.

17. Authors' Biographies

Dr. David R. Pratt, an Assistant Professor in the Department of Computer Science at the Naval Postgraduate School. He is extremely active in the DIS community as both a developer and information resource for systems development and network integration.

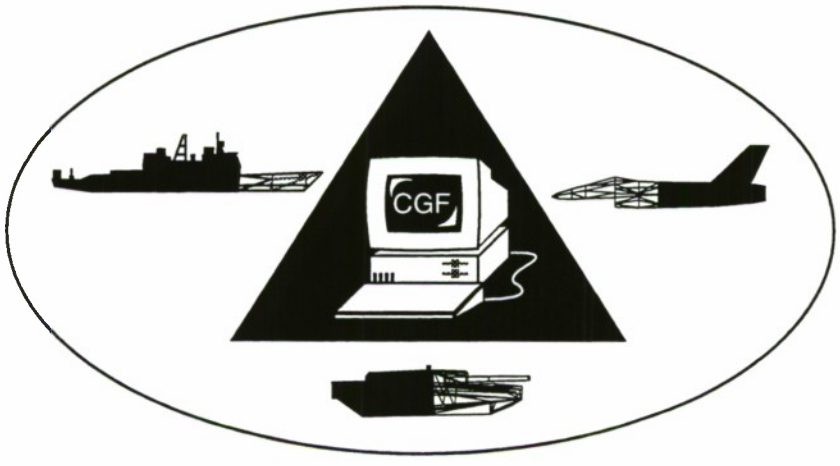
Maj. Gary MacAndrews, USA, earned his Masters in Computer Science from the Naval Postgraduate School in September, 1994. A former Armor Company Commanding Officer, he is now an Aqusition Corps Officer.

Dr. Robert B. McGhee is a full Professor in the Department of Computer Science of the Naval Postgraduate School. A Fellow of the IEEE, he lead the Ohio State Walking Machine project. His current interests include dynamics and autonomous entity interaction.


CGF

Session 5b: Route Planning I

Campbell, SAIC
Karr, UCF/IST
Hoff, Hughes Research Laboratories


CGF

Route Planning in CCTT

Chuck Campbell, Richard Hull, Eric Root, Lance Jackson
Science Applications International Corporation
3045 Technology Parkway
Orlando, FL 32826
{campbell hull eroot jackson}@greatwall.cctt.com
(407) 282-6700

1. Abstract

One aspect of Computer Generated Forces (CGF) terrain reasoning is the ability to get simulated entities from one location to another. The first step in this task usually involves some level of planning to generate a path between locations for the entity to follow. In the Close Combat Tactical Trainer (CCTT) CGF system, this planning is accomplished through three major terrain reasoning areas: routing, obstacle avoidance, and dynamic entity avoidance. Routing provides a high-level path, or route, around large terrain features such as rivers, forests, and urban areas as well as routes which follow road networks that criss-cross the terrain database. Obstacle avoidance, on the other hand, provides a low-level path through the database, avoiding all features considered as obstacles, including individual trees and buildings. Dynamic entity avoidance predicts potential collisions between entities and takes appropriate actions to avoid contact. This paper focuses on the implementation of obstacle avoidance and road route planning for the CCTT CGF.

2. Introduction

The CCTT system is the first trainer in the Combined Arms Tactical Trainer (CATT) family of training systems. CCTT is a real-time networked simulation environment designed to provide training of specific military skills at a fraction of the cost of an equivalent field exercise. CCTT is composed of several different types of systems including Manned Modules, Workstations, and Semi-Automated Forces (SAF). Manned Modules consist of crew cabin simulators, including M1A1s, M2A2s, HMMWVs, and dismounted infantry (DI). Workstations provide simulation capabilities for the battalion support staff, After Action Review, and simulation support. SAF provides additional friendly (BLUFOR) and enemy (OPFOR) entities by emulation of vehicle dynamics and crew behaviors. Each of these systems communicates using the Distributed Interactive Simulation (DIS) network protocol. A comprehensive discussion of terrain reasoning within CCTT can be found in (Watkins 1995).

This paper describes the work that has been done in CCTT to support road routing and low level obstacle avoidance. Discussion of cross-country route planning will not be presented here. Section 3 discusses what is involved in road routing and how it is used by the user. Section 4 describes the iterative-deepening A* algorithm used to find optimal road routes. Preliminary results for the road route planner are given in Section 5. Section 6 discusses obstacle and dynamic entity avoidance. Section 7 describes relocatable objects that affect obstacle avoidance and road routing. Section 8 presents related work. Section 9 gives the authors' plans for future work.

3. Routing Across Networks

A functional requirement of CCTT is the

ability to route vehicles over road networks that criss-cross the simulation database. This presents several challenges to the terrain reasoning system due to the complexity of the problem of finding optimal routes within a database with the size and feature density of the CCTT databases. An additional consideration is dynamic features, such as destructible bridges and log cribs, which can change the road network at run-time. CCTT's approach combines specialized representations of the road network with an efficient algorithm that manipulates them. These representations are computed in an *a priori* compilation phase, which is performed off-line thereby reducing the level of computational effort required during the simulation. Moreover, road routing is implemented as a separate process which allows the caller to perform other computations while the route is being generated.

3.1 Usage

The road routing functionality provided in CCTT is relatively simple to use. The user supplies a proposed route that consists of a series of points that the user wants fully expanded into a road route, and a road snapping threshold. The road snapping threshold is used to determine how far from an actual road the point can be before it is considered to be non-road point. The user simply sends a request to the routing process and then periodically checks to see if the plan has finished. Using a separate process allows the user to continue with other tasks and allows the routing process to take as much time as necessary to completely expand the route. Once the routing process has finished expanding the route, it returns the completed route to the user with any error conditions set. Currently several things can go wrong when generating a road route:

- A point specified by the user is off the database.
- Point(s) marked as road points with no corresponding road in the database within the road snapping threshold.
- During the route expansion a point the user specified may be unreachable. That is, the road network may be unconnected or obstacles may prevent reaching the goal.

When an error occurs during route expansion the partially completed route is returned to the user along with the bad point. The user may then fix the problem and resubmit the route for further expansion.

4. IDA*

The algorithm used to search the road network, and which produces optimal road routes is a variation of the A* (Hart et al. 1968) algorithm known as iterative-deepening A* (IDA*) (Korf 1985). A* is a popular search algorithm that improves upon branch-and-bound techniques by incorporating dynamic programming and an admissible estimator of the remaining distance to the goal location called a heuristic estimate. A heuristic estimate is considered admissible if it is guaranteed to always underestimate the remaining distance. Straight-line distance is commonly used as such a heuristic, because no route can ever be shorter than the straight-line distance. One disadvantage of the A* algorithm is its need to save all partial paths to the goal for "possible" expansion. These partial paths are usually stored in a sorted queue, whose size can become extremely large. This may severely impact run-time memory requirements resulting in memory allocation failures.

IDA* is a variation of A* that resolves much of the run-time memory problem by eliminating the need to maintain the sorted queue of partial paths. Instead, this algorithm searches a tree or graph by repeatedly performing depth-first searches to greater and greater depths, hence the name "iterative-deepening". This might seem to contradict one's intuition in that the algorithm examines most of the same nodes over and over again. However, it is has been proven that as the branching factor of the tree increases, the performance of IDA* becomes

asymptotically close to the performance of A* (Korf 1985). Therefore, the performance of IDA* is not as bad as it may appear to be.

4.1 Space vs. Time Tradeoff

Given that IDA* is never faster than A*, when should it be used? The answer to this question depends on whether the critical aspect one is interested in is space or time. In this particular application, where the planning of road routes is primarily performed pre-exercise, the time factor is not as important as run-time memory requirements. If on the other hand, time is crucial and memory is not a concern, then A* should be used. An interesting comparison of the space vs. time tradeoff between A* and IDA* for cross country route planning can be found in (Karr et al. 1995).

4.2 Trees and Graphs

The topology of the search space is another issue that one must be prepared to face. Road networks represent a graph topology and must be handled differently from tree topologies. For example, during a depth-first search the algorithm should not visit a given intersection more than once, i.e., care must be taken to prevent cycles. This is accomplished by maintaining a "visited" boolean for each intersection which is made true when the intersection is expanded, and is reset to false after the recursive call has terminated. Additionally, when using IDA* on graphs it is very important to remember the cost of the least cost path from the source to each intersection that has been visited. This dynamic programming principle saves a tremendous amount of time because many paths of the depth-first search can be pruned.

4.3 Algorithm

CCTT's implementation of IDA* for road routing is embodied in an ADA procedure called *Find_Route*. *Find_Route* takes two intersections, the source and the goal, and returns a list of road segments defining the route if one exists. The algorithm is shown in Figure 1.

The depth-first search routine recursively traverses the graph by expanding all paths from the source until the cost of that particular path is greater than the current threshold cost. If the goal is reached along a particular path, the cost of the path is compared against that of any other solutions that might have been found and if the cost of the newly found path is less than the best so far, we save the new path and discard the previous best. The procedure *Depth_First* is shown in Figure 2.

5. Road Routing Observations

Preliminary investigations have shown that the iterative-deepening A* algorithm is working well during pre-exercise road route planning. The road routing process generates distance-optimal routes in an "acceptable " amount of time. Because integration activities are under way, however, we cannot conduct system level timing tests of the road routing capability at this time.

Figure 3 shows a sample of the road networks found in a preliminary release of the CCTT "Primary1" terrain database. The area shown represents approximately 2 square kilometers. The dark lines are the linear features which represent the road network.

Figure 4 shows an example of a route across the network. Although the route appears to be sub-optimal, it is the shortest route which visits all of the route points selected by the user.

6. Obstacle Avoidance

CCTT CGF obstacle avoidance performs short term routing around individual terrain features such as trees and buildings. It uses the long term route information, to be provided by CCTT SAF's cross country routing, as a guide.

Find_Route (source, goal)

1. Initialize Threshold_Cost to the straight-line distance from source to goal.
2. Initialize final path array.
3. While not Goal_Found and Threshold_Cost does not equal Infinite_Value loop

   3.1 Set Next_Threshold to Infinite_Value

   3.2 Set the initial cost to 0.

   3.3 Perform depth-first search from source until the cost of all paths exceed Threshold_Cost.

   3.4 If not Goal_Found then Threshold_Cost = Next_Threshold.

end loop

4.0 If Threshold_Cost = Infinite_Value then Success = False,
else Return the list of segments defining the route.

Figure 1: Find_Route Algorithm



Figure 3: Primary1 Road Network



Figure 4: An Example Road Route

The long term route is divided into smaller segments along which obstacle avoidance is performed. Obstacle avoidance provides a list of path points which define a clear path through the obstacles along the short term route.

CCTT CGF short term route planning is accomplished through a combination of static and dynamic obstacle avoidance. The separation of static and dynamic obstacle avoidance for CCTT CGF was influenced by several factors:

- A clear path through static obstacles can be accurately computed. This path will not need to be altered unless a dynamic

Depth_First (root, goal, threshold cost, actual cost)

1. Calculate the heuristic cost of the root intersection as the sum of the actual cost from the source to the root and the heurstic estimate of the cost from the root to the goal.
2. If the heuristic cost is greater than the threshold cost then
   2.1 If the heuristic cost is less than the next threshold then save the heuristic cost of the root as the new next threshold
   Else
   2.2 Add the root intersection to the current path.
   2.3 If the root = the goal then
      2.3.1 If the cost of this path is the cheapest so far then save it
      Else
      2.3.2 For each intersection connected to root loop:
         2.3.2.1 If the segment connecting intersection(i) to root is trafficable, and intersection(i) has not been visited, and the cost from the source to the root is the least we've encountered, then
            2.3.2.1.1 Calculate the actual cost for intersection(i)
            2.3.2.1.2 Mark intersection(i) as visited.
            2.3.2.1.3 Recursively call Depth_First(intersection(i) , goal, threshold cost, actual cost)
            2.3.2.1.4 Mark intersection(i) as unvisited.

Figure 2: Depth_First Algorithm

entity or relocatable object forces a replan. It is assumed that most potential conflicts with dynamic entities can be avoided through speed changes.

- It is assumed that CCTT database feature densities will often render extrapolations of entity positions useless after some tens of meters, due to frequent heading modifications.

- Any attempt at short term route following must continue to make frequent checks for dynamic entities in the area, since the behavior of a dynamic entity over time cannot be accurately determined.

- Road following will only need static obstacle avoidance when an entity must leave the road to avoid a collision, which is assumed to be infrequent, while dynamic entity avoidance is constantly queried.

As with most terrain reasoning functions, obstacle avoidance is a computationally expensive operation. The approach taken by CCTT CGF is based upon ModSAF's libmovemap routines.

6.1 Static Obstacle Avoidance

The static obstacle avoidance algorithm used in CCTT CGF is heavily based upon the method in ModSAF for generating spatial curves (Smith 1994). This approach generates Catmull-Rom splines (Foley et al. 1990) to produce a smooth course through the database. The control points for the splines are taken from the vehicle's location and heading, the long term route points generated by cross country routing, and from skirt points generated to avoid static obstacles. A linear approximation to the splines is computed to derive the points which make up the obstacle avoidance path.

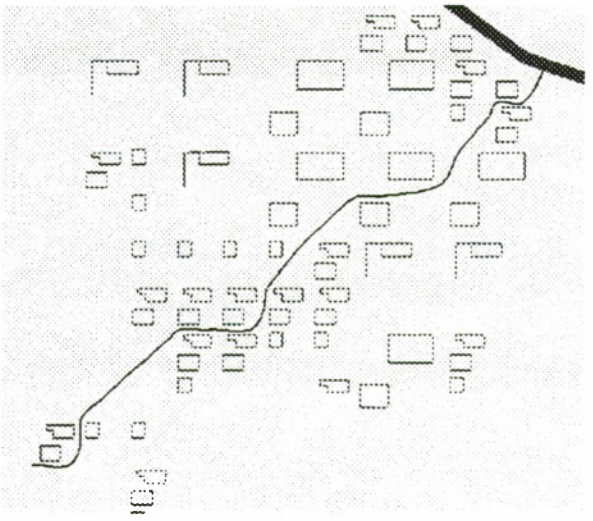


Figure 5: Example short term route path generated around buildings

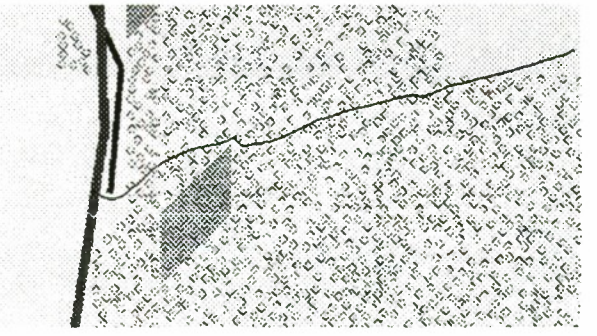


Figure 6: Example short term route path generated around trees, steep slope areas, and rivers

Preliminary results are shown in Figures 5 and 6.

The CCTT obstacle avoidance algorithm consists of two main units. The first unit examines the overall route supplied by the long term cross country router and determines the portion of the route to be currently considered by obstacle avoidance. Using the moving entity's position and the route corridor, the relevant obstacles in the area around the route are extracted into an obstacle map. The obstacles in the map are expanded by the width of the routing entity so that a simple line intersection can be used to detect collisions.

The second unit creates the initial path from the entity's location and heading and the long term route points within the short term planning horizon. If the end of the route is not reached before the short term horizon, an additional point is added at the edge of the obstacle map to ensure the entity proceeds correctly to the next set of route points.

Once the initial path is determined, a recursive procedure checks the current path against the obstacles in the obstacle map. If an intersection with an obstacle is found, the routine attempts to go around the obstacle on both sides by generating skirt points in between the current path points. If the obstacle can be skirted, the routine recursively checks the new path, attempting to skirt the obstacle in the direction with the shortest skirt path first. If no skirt paths can be found around an obstacle, a failure flag is set and the recursive algorithm backs up and attempts a different path.

Once a path with no obstacle intersections is found, it is optimized by removing all skirt points which are no longer important. Skirt points required to get around an early obstacle are often no longer needed once a later obstacle is skirted. The resulting successful path is evaluated for complexity, with greater numbers of control points and longer paths considered more complex, and the least complex path found so far is stored. After a predetermined number of successful paths are found, the recursion is stopped and the best path found is returned.

A detailed discussion of the methods used to expand the obstacles in the map, to compute the spline coefficients, and to generate the skirt points around obstacles can be found in (Smith 1994).

6.2 Dynamic Entity Avoidance

The dynamic entity avoidance in CCTT CGF also uses ideas implemented in ModSAF. Dynamic entity avoidance must be checked frequently, since the headings and velocities of entities are constantly changing.

The algorithm projects an entity along its static obstacle path at small time intervals to some event horizon. Concurrently, the projected positions of other entities in the area are calculated, using their velocities and the time interval. An entity does not consider other entities currently behind him; it is assumed that the trailing entity will avoid the leading entity.

The bounding boxes of all entities are projected into the xy plane and are checked for overlap with the detecting entity Figure 7. Non-overlaps are thrown out, and overlaps are further investigated using a more rigorous check to determine if the entities will in fact collide. When a probable collision is detected, the available information (the time until collision, entity relative velocities, etc.) is used to determine the appropriate action to take.

An interesting problem arises when the courses of two entities converge but do not cross, or when entities that are following similar paths have different speeds. In these situations, collisions can be difficult to anticipate. Entities traveling in formation are particularly susceptible to these problems; swerving to avoid a tree or slowing to avoid an entity can result in a collision with another entity in one's own platoon.

Since CCTT requires large numbers of entities to travel in formation, an efficient solution had to be found for these situations. Once the bounding box check described above has indicated an overlap, then a check is made to determine if the two entities actually come close enough to each other to cause a collision.

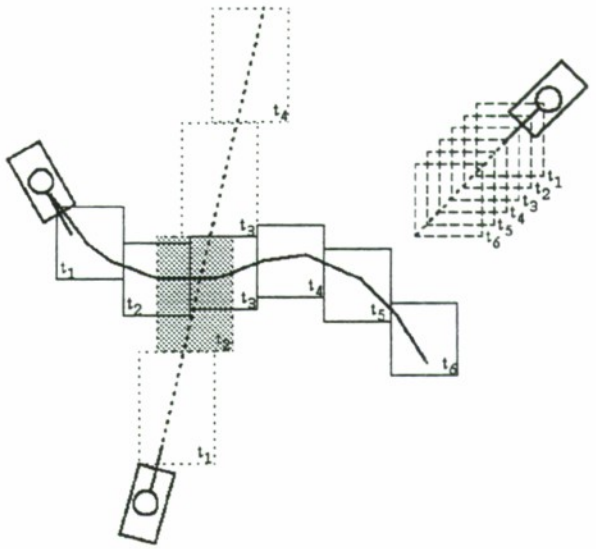


Figure 7: Detecting possible collisions between entities by projecting the areas entities cover during discrete time steps

By describing the motion of both entities with parametric equations, the time when the entities come within a given distance of each other can be determined.

Given two entities with velocities V_1 and V_2 and locations L_1 and L_2, the two dimensional distance d between them at any given time t is shown in Figure 8.

By setting the distance to the collision distance threshold desired, and by scaling the velocity vectors over the required time frame, a probable collision can be accurately detected.

7. Relocatable Objects

Relocatable objects are terrain features which can be dynamically placed into the terrain database during a simulation. In CCTT, Combat Engineering entities place relocatable objects to provide mobility, countermobility, and survivability.

Relocatable objects may be surface obstacles, changes to terrain geometry, or other traffica-

$$\mathrm{d} = \sqrt{(V_{1_x}t - L_{1_x} - V_{2_x}t - L_{2_x})^2 + (V_{1_y}t - L_{1_y} - V_{2_y}t - L_{2_y})^2}$$

Squaring both sides and simplifying:

$$d^2 = [(V_{1_x} - V_{2_x})^2 + (V_{1_y} - V_{2_y})^2]t^2 + 2[(V_{1_x} - V_{2_x})(L_{1_x} - L_{2_x}) + (V_{1_y} - V_{2_y})(L_{1_y} - L_{2_y})]t + (L_{1_x} - L_{2_x})^2 + (L_{1_y} - L_{2_y})^2$$

Defining the relative velocity and relative location vectors as:

$$V_{rel} = V_1 - V_2$$

$$L_{rel} = L_1 - L_2$$

We have:

$$d^2 = (V_{rel} \cdot V_{rel})t^2 + 2(V_{rel} \cdot L_{rel})t + (L_{rel} \cdot L_{rel})$$

Subtracting d^2 from both sides and using the quadratic formula:

$$t = \frac{-V_{rel} \cdot L_{rel} \pm \sqrt{(V_{rel} \cdot L_{rel})^2 - (V_{rel} \cdot V_{rel})(L_{rel} \cdot L_{rel}) + (V_{rel} \cdot V_{rel})d^2}}{V_{rel} \cdot V_{rel}}$$

Since $\|V_{rel} \times L_{rel}\|^2 = \|V_{rel}\|^2\|L_{rel}\|^2 - \|V_{rel} \cdot L_{rel}\|^2$:

$$t = \frac{-V_{rel} \cdot L_{rel} \pm \sqrt{d^2(V_{rel} \cdot V_{rel}) - \|V_{rel} \times L_{rel}\|^2}}{V_{rel} \cdot V_{rel}}$$

Figure 8: Derivation of equations to determine the time two when two vehicles come within a certain distance of each other.

bility factors. The list of relocatable objects used for CCTT includes log crib, abatis, tank ditch, concertina wire, minefield, armored vehicle launched bridge (AVLB), ribbon bridge, and vehicle and DI prepared fighting positions. Each of the relocatable objects can be damaged, destroyed, or breached.

For the most part, relocatable objects can be treated as any other terrain obstacles to be avoided. The interesting behavior for obstacle avoidance is associated with the bridges, minefields, and prepared fighting positions.

Most terrain features are either treated as obstacles to be avoided (e.g. buildings) or are ignored (e.g. roads) with respect to obstacle avoidance. Relocatable bridges, however, present a special case, in that they are features to be approached, rather than avoided. Obstacle avoidance must recognize these features as trafficable and make use of them where appropriate.

Minefields present several interesting cases. An opposing force may not know about an existing minefield, in which case the minefield is not an obstacle until its presence is known. When the presence of a minefield is known, it becomes an obstacle, unless it is breached. Route planning, both at the high and low levels, must be aware of and take advantage of breached lanes in the minefield when planning trafficable paths.

Prepared fighting positions, such as hull defilades, are generally considered obstacles for short term routing. However, obstacle avoidance must be able to plan paths into prepared positions for entities wishing to take cover in them.

Relocatable objects also affect road route planning. The approach we are currently investigating is to change the road network at run-time when relocatables are placed. For example, if a log crib or abatis were placed across a road segment, the road network would be modified to reflect the obstacles affect on trafficability. The original road segment is split into smaller segments by creating two intersections that mark the extents of the obstacle. The road segment defined by these two intersections is not trafficable, and its associated trafficability flag is set accordingly. The trafficability of the new road segments leading up to and away from the obstacle are not affected. This allows vehicles to be routed right up to the obstacle if necessary. Moreover, the changes to the road network are localized and can be made quickly.

8. Related Work

While research in route planning for CGF purposes is relatively new, a great deal of work has been done in the field of robotics regarding motion planning (Arkin 1989, Brooks 1986, Chattergy 1985, Thorpe 1984). An excellent compendium of route planning techniques and their associated complexities can be found in (Mitchell 1988). A detailed discussion of "least-risk" watchman routes, which is extremely relevant to CGF researchers interested in developing systems capable of generating covered and concealed routes can be found in (Gewali et al. 1989). The artificial intelligence community has also given attention to this problem (Kuipers and Levitt 1988, Levitt and Lawton 1990, McDermott and Davis 1984). A recent, knowledge-based approach to the problem of finding "reasonable" road routes is discussed in (Goel et al. 1991). Our approach to planning road routes and performing obstacle avoidance was directly influenced by the techniques included in ModSAF (Smith 1993). However, there are several noticeable differences. IDA* was used instead of A* for planning road routes and route planning is implemented as a separate process.

9. Future Work

This paper describes the status of the CCTT routing capability as of March 1995. Future work will focus on the implementation of cross country routing and enhancing current routing capability.

For cross country routing, two approaches are currently being investigated. One is the algorithm used by ModSAF. The other is an abstraction of the obstacle avoidance routines which will operate on larger features such as forests and urban areas, ignoring individual trees and buildings.

Several enhancements to existing routing functionality are being considered. CCTT routing is currently performed using only shortest distance as a heuristic. More sophisticated heuristics for choosing routes are being explored, such as modifying segment weights by factoring in terrain slope and soil type (including "wet" soils). Obstacle avoidance and route planning enhancements may include a more sophisticated path optimizing (smoothing) algorithm, reuse of obstacle maps for multiple entities within a platoon, and support for multi-level terrain and relocatable objects.

10. References

Arkin, R. Navigational Path Planning for a Vision-Based Mobile Robot. *Robotica*, 7:49–63, 1989.

Brooks, R. A Robust Layered Control System for a Mobile Robot. *IEEE J. of Robotics and Automation*, RA-2(1):14–23, 1986.

Chattergy, R. Some Heuristics for the Navigation of a Robot. *Int. J. of Robotics Research*, 4(1):59–66, 1985.

Foley, James D., van Dam, Andreas, Feiner, Steven K., and Hughes, John F. *Computer Graphics Principles and Practice, Second Edition.* Addison-Wesley Publishing Company, Inc., 1990.

Gewali, L., Meng, A., Mitchell, J.S.B., and Ntafos, S. Path Planning in 0/1/infinite Weighted Regions with Applications. Technical Report 831, Cornell University, College of Engineering, 1989.

Goel, Ashok K., Callantine, Todd J., Shankar, Murali, and Chandrasekaran, B. Representation, Organization, and Use of Topographic Models of Physical Spaces for Route Planning. In *Proceedings of the 7th Conference on Artificial Intelligence Applications CAIA-92*, pages 308–314. IEEE, 1991.

Hart, Peter E., Nilsson, Nils J., and Raphael, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, 4(2), 1968.

Karr, Clark R., Rajput, Sumeet, and Breneman, Larry. Comparison of A* and Iterative Deepening A* to Graph Search. In *Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation.* Institute for Simulation and Training, 1995.

Korf, Richard E. Iterative-Deepening-A*: An Optimal Admissible Tree Search. In *IJCAI-85.* Morgan Kaufmann, 1985.

Kuipers, B. and Levitt, T. Navigation and Mapping in Large-Scale Space. *AI Magazine*, 9(2):25–43, 1988.

Levitt, T. and Lawton, D. Qualitative Navigation for Mobile Robots. *Artificial Intelligence*, 39, 1990.

McDermott, Drew and Davis, Ernest. Planning Routes through Uncertain Territory. *Artificial Intelligence*, 22:107–156, 1984.

Mitchell, J.S.B. An Algorithmic Approach to Some Problems in Terrain Navigation. *Artificial Intelligence*, 37:171–201, 1988.

Smith, Joshua E. ModSAF programmer's guide. Technical report, Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1993.

Smith, Joshua E. Near-term movement control in ModSAF. In *Proceedings of the 4th Conference on Computer Generated Forces and Behavioral Representation.* Institute for Simulation and Training, 1994.

Thorpe, C. Path Relaxation: Path Planning for a Mobile Robot. Technical Report CMU-RI-TR-84-5, CMU Robotics Institute, 1984.

Watkins, Jon. Terrain Capabilities in CCTT. In *Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation.* Institute for Simulation and Training, 1995.

11. Biographies

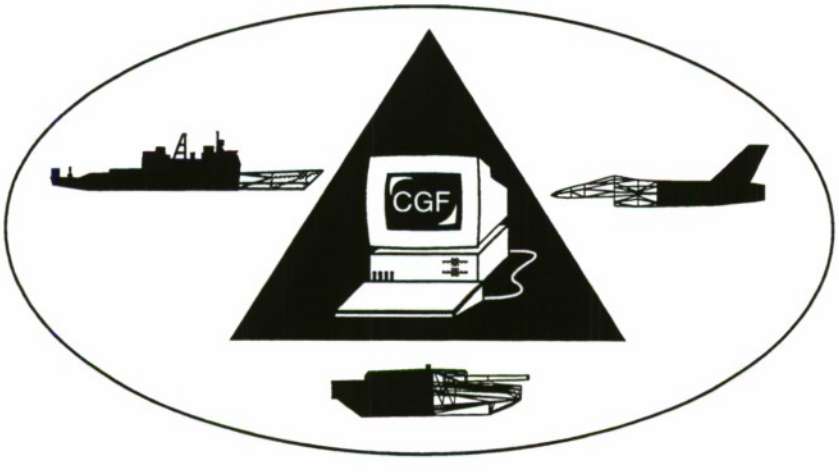
Chuck Campbell is a Software Engineer on the CCTT project at SAIC in Orlando, Florida. He has earned a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from Indiana University. Mr. Campbell has over four years' experience developing Computer Generated Forces software. His interests include simulation, graphics, and computational geometry.

Richard Hull is a Software Engineer on the CCTT project at SAIC. He is also a PhD student at the University of Central Florida, where he is working in the areas of Natural Language Understanding and Knowledge Acquisition. He has four years of experience implementing various components of CGF systems. His related interests include the application of artificial intelligence techniques to CGF problems, high-level planning, and knowledge representation.

Eric Root is a Software Engineer on the CCTT project at SAIC. He has worked in the simulation and training industry for over two years. Mr. Root has earned a B.S. in Computer Science and Mathematics from Missouri Western State College; he is currently an M.S. student

in Computer Science at the University of Central Florida.

Lance Jackson is currently a student at the University of Central Florida working towards a B.S. in Computer Science. His research interests are simulation and computational geometry.


CGF

Dynamic Obstacle Avoidance for Computer Generated Forces

Clark R. Karr, Michael A. Craft, and Jaime E. Cisneros
Institute for Simulation and Training
3280 Progress Drive, Orlando, Florida 32826
ckarr@ist.ucf.edu

1. ABSTRACT

Techniques to allow simulated entities to avoid static terrain, such as trees, buildings, rivers, etc., have been in use in Distributed Interactive Simulation (DIS) environments for many years. Avoidance of objects in motion, "dynamic obstacles", is a complicated issue. Although simple dynamic obstacle collision avoidance has been implemented, the resulting behavior is usually less than realistic. While complex algorithms for dynamic obstacle collision avoidance are computationally expensive. The paper presents a novel approach that allows simulated DIS entities to make reasonably intelligent and realistic maneuvers to avoid dynamic (and static) objects without excessive computational cost. This Dynamic Obstacle Avoidance (DOA) Model combines two disparate motion planning approaches: potential fields and grid based route planning.

2. INTRODUCTION

2.1 General Path Planning Approaches

Motion planning with particular emphasis on robot path planning and robot manipulator path planning has seen considerable work (Hwang et. al. 1992). There are four broad categories of path planning approaches: free/blocked space analysis, vertex graphs analysis, potential fields, and grid (regular tesselation) based algorithms (Thorpe 1984). Each approach has strengths and weaknesses.

2.1.1 Free/Blocked Space Analysis

In a free space approach, only the space not blocked or occupied by obstacles is represented and planning occurs within this space. For example, using Voronoi diagrams (Roos and Noltemeier 1984) to represent the center of movement corridors is an efficient free space approach. However, Voronoi diagrams and other free space approaches have some deficiencies. First, they tend to generate unrealistic paths. Paths derived from Voronoi diagrams follow the center of corridors while paths derived from visibility graphs (Mitchell 1988) clip the edges of obstacles. Second, the width and trafficability of corridors are typically ignored. Third, distance is generally the only factor considered in choosing the optimal path.

In contrast, near-term maneuver control in ModSAF is a blocked space analysis using analytic geometry and AI search (Smith, 1994). Obstacles are projected in time and routes are found outside the obstacles by curve fitting through spline control point adjustment. The route finding algorithm is reapplied throughout movement to recalculate and refine the current route segment. The speed, acceleration, and turn rates are determined for the entire route segment. Although the computational expense of this approach is controlled by scheduling planning events as infrequently as possible, this approach is computationally wasteful in that carefully crafted routes are discarded and recalculated in exactly those situations where dynamic obstacle avoidance is needed most.

2.1.2 Vertex Graph Analysis

In the vertex graph approach, only the endpoints (vertices) of possible path segments are represented (Mitchell 1988). This approach has three difficulties. First, i is suitable only for spaces that have a sufficient density of obstacles; determining the vertices in "open" terrain is difficult. Representing only path vertices creates two other difficulties. Second, trafficability over the path segments is not represented; routes segments between arbitrary vertices are typically "open" or "blocked". Third, factors other than distance can not be included in evaluating possible routes. In the military simulation domain, concealment and cover are important factors in route planning.

2.1.3 Potential Fields

In the potential field approach, the goal (destination) is represented as an "attractor", obstacles are represented by "repellors", and the vehicle is pulled toward the goal while being repelled from the obstacles (NASA 1991). There are two difficulties with the potential field approach. First, the vehicles can be attracted into box canyons from which they can not escape (NASA 1991). Second, some elements of the terrain may simultaneously attract and repel. For example, an obstacle to movement, a

repellor, may create an area of concealment. A vehicle should be attracted to the obstacle for concealment while being repelled from the obstacle creating the "visibility shadow".

2.1.4 Regular Grids

In the regular grid approach, a grid overlays the terrain, terrain features are abstracted into the grid, and the grid rather than the terrain is analyzed. Each grid cell is typically marked as "open" or "blocked". Quadtrees are an example of the regular grid approach (Mitchell 1988). This approach simplifies the analysis but has two disadvantages. First, "jagged" paths are produced because movement out of a grid cell is restricted its "n" neighbors (typically four or eight for square grids and six for hexagonal grids). Second, the grid granularity (size of the grid cells) determines the smallest "opening" between obstacles that can be identified. When the granularity is too large, small openings in obstacles (e.g. bridges over rivers) are lost. Increasing the grid granularity to capture the small openings increases the computational expense of the analysis.

3. THE DOA MODEL

The DOA Model combines the potential field and regular grid approaches into a single mechanism for avoiding moving and static obstacles during movement along a predetermined route. The Neural Net paradigm (Glasius et. al. 1994 and Hertz et. al. 1992), was the genesis for this work, although the traditional AI hill climbing is another excellent metaphor (Charniak and McDermott 1987).

A "small" rectangular grid overlays the vehicle and the area to its front and sides. The cells in the grid fall into several classes:

1. barrier: a static obstacle the vehicle cannot cross and which does not change position, such as a river,
2. target: represents the position the vehicle wants to reach (potentially interpolated from the next route point),
3. vehicle: represents the position of the vehicle doing the routing,
4. obstacle: represents the positions and projected positions of moving objects, and
5. open: all other cells.

For example:

	A	B	C	D	E	F
1			B			
2			B			
3			B			
4						T
5	V			O		
6				O		
7						
8						

Figure 1 : DOA Grid.

In this example, the vehicle has no barriers (B) between it and its target, but a moving obstacle (O) is, at least temporarily, blocking a line from the vehicle (V) to the target (T).

3.1 Application of the DOA Grids

This work was performed in the Institute for Simulation (IST) Computer Generated Forces (CGF) Testbed (Smith 199?). When a CGF vehicle is to route to a destination, a long term planner selects a collection of intermediate points (route points) to be used in traveling to the destination. The long term planner analyses only static terrain features. To avoid moving objects, the DOA logic is applied periodically during route transversal. The frequency of DOA analysis is determined from the vehicle's speed such that the vehicle will move approximately 1 cell before the next DOA analysis The grid is laid out so that the vehicle is on an edge and the target, or an interpolated target, falls on the opposite edge (see Figure 4).

3.2 Interpretations of the DOA Grid

The grid can be viewed as a network of cell connections. For example, consider a subset of the full graph showing a path from the vehicle in Figure 1 to its target destination:

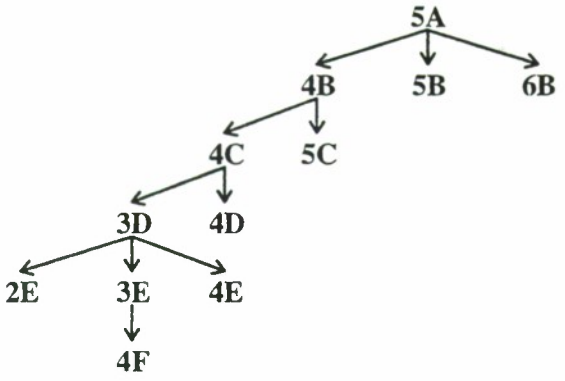


Figure 2 : Path as Tree Transversal.

It is the directed graph view that triggered the IST notion of looking for parallels between this problem and the neural net and potential field approaches. Traditionally, such a graph would be used to reduce the problem to that of a search avoiding "danger areas" (Cortes-Rello and Golshani 1990). However, the search metaphor inevitably leads one to treat grid cells as open (passable) or closed (blocked) (Mitchell 1988); something more subtle is needed. While cell 4D is not blocked, its proximity to a moving obstacle should make 4D less attractive than 3D (hence the choice of 3D rather than 4D in Figure 1). It is not obvious how to take such things into account when viewing this as a simple search problem.

3.2.1 The Neural Net Model

If we view the cells as elements in a Hopfield neural net (Hertz 1992), the cells (particularly 3D and 4D in the example at hand) should be influenced by their adjoining cells. In neural net theory, the "energy" of a cell is calculated in two steps. First, the contributions of neighboring cells are summed:

$$h_j = \sum w_{ij} h_i$$

Where h_j represents the energy of a cell j, w_{ij} represents the "connection weight" from cell i to cell j, and h_i represents the energy of cell i. The second step involves "clamping" h_i within a 0/1 or ±1 range with an "activation" function, g(h). Neural net theory provides two frequently used activation functions: a sigmoid function and a hyperbolic function. However, Glasius (1994) suggests that a linear activation function, g(x) = Kx, is adequate for propagating the obstacle values within the net (where K is a constant in the range [0, 1.25)). In this research, the linear activation function was used and the range expanded to [0,10,000). Understandably, large K values tended to magnify the effects of obstacles.

The DOA Grid can be viewed as a neural net with connection weights held at 1. Thus, only cell values are manipulated.

3.2.2 The Potential Field Model

In the DOA Model, the grid and its obstacles are manipulated via the potential field metaphor. The DOA algorithm assigns the target cell a negative (attractive) potential, the barrier cells potentials of 0, and the obstacles cells positive (repulsive) potentials. Neural Net mechanics propagate the potentials throughout the grid. Thus, a cell's value represents the combined attractive and the repulsive potentials of the target, barriers, and obstacles. For simplicity, one can view these values as either elevations or temperatures. If viewed as elevations, the vehicle is seeking the lowest point not unlike a marble rolling downhill. If viewed as temperatures on a uniform sheet (the grid), the vehicle is trying to negotiate the sheet to reach the coolest point.

As stated, the problem still appears to be a simple search: some points are forbidden, there is a start and a finish location. However, if we view the sheet as heat conducting, and the obstacles as points where heat is being applied (perhaps with a soldering gun) and the target as a point being cooled, it becomes clear how one cell can influence those around it. The key now is to let time pass so the effects of the heating and cooling can spread. After some time we might have a new situation such as:

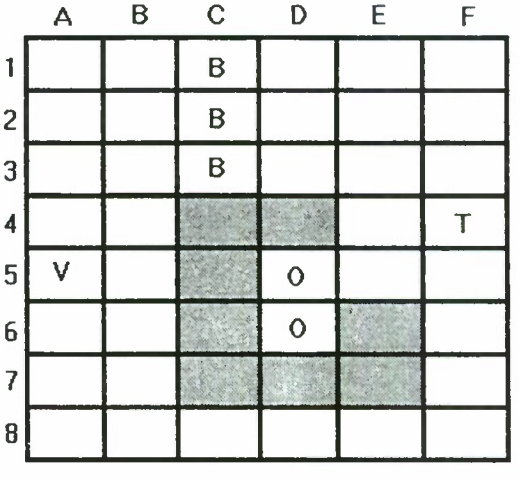


Figure 3 : Heated DOA Grid.

The shaded squares indicate an elevated temperature because the obstacles have heated the cells next to them. Eventually, those cells could warm those next to them, and so on. Some cells (such as 5C) are adjacent to 2 obstacle cells and so are heated from two sources, others, such as 4E are heated on one side and cooled on the other. Rather than attempt to apply any realistic temperature model, discrete time steps are taken and at each step a cell's new temperature is a weighted average of its own temperature and the temperature of the surrounding (or, if viewed as a graph, the connected) cells' temperatures. At each iteration, a new set of cells are affected by obstacle cells and cells which had already been adjusted are re-adjusted.

If we view the DOA Grid through the elevation metaphor, the initial grid is flat but for some plateaus (obstacles) and a pit (the target). The smoothing process causes the ground (cells) around the poles and pits to erupt or sink. Continued iterations cause the grid to approach terrain sloping smoothly from the vehicle toward the goal and around the "hills" surrounding the obstacles and barriers. The path to be taken avoids high spots and runs down to the low areas.

The resulting grid captures a great deal of information and is the basis for the algorithms developed. A cell containing an obstacle effects surrounding cells and the resulting gradient is an imprecise measure of the probability the vehicle will move there. A gradient based search would now select 3D rather than 4D. It is quite easy now for people to express opinions with a glance at the grid. The path already described does not go too much out of the way but avoids 1 "hot" square. A path starting out due South East could stay on cool squares for the entire trip, albeit by taking a longer trip.

A central premise of this work is that a grid can be constructed, smoothed (which means carrying out several iterations to allow the potentials to spread), and an algorithm to select velocity applied very rapidly.

The grid approach is prone to all of the classic hill climbing problems. Our view is "upside down", we can get trapped in local minimum rather than a local maximum; in any case, local extremes are a problem (Charniak and McDermott, 1987). To some extent, these problems are mitigated by the fact that there is only one low point at the outset; if a cell is depressed, there must be a clear path from that cell to the target because it was influenced by the target. In general, a lower point is either further from obstacles or closer to the target.

3.3 Long Range, Short Range, and Immediate Planning

Planning a complete route is a long range process. What direction to travel for the next few seconds is dependent on the long term goal and the local conditions. Given arbitrary computation time, long term planning can be used for DOA problems by treating the moving obstacles as fixed for the purpose of computation, and re-computing frequently with updated obstacle positions. Unfortunately, long term planning probably requires more effort than can be expended for rapid speed and course adjustments. In contrast, considering only the immediate vicinity (immmediate planning) can avoid collisions by making "snap decisions" on what to do next without full analysis of the situation (in particular, without guaranteeing that the destination can be reached from the new position).

3.3.1 Short Range Planning

The A* search algorithm (Winston 1992), is a efficient long range search algorithm. In the DOA Model, A* treats the DOA grid as a terrain map where, essentially, the distance traveled is minimized (taking the elevated areas into account by charging extra distance for crossing elevated areas). Because the DOA grid is small and provides only local information, the route found by A* is not a long range route to a destination or intermediate route point. Rather it is a short range route around and through the local obstacles.

3.3.2 Immediate (Next Step) Planning Algorithms

The other algorithms tested are of the immediate planning type (Karr, 1995). Only the cells adjacent to the vehicle are considered in making course and speed adjustments. The danger to such an approach, of course, is that the vehicle could become trapped or cycle. These concerns are greatly mitigated by the way the method is applied. First, each cell's value reflects the combined potentials within the entire DOA grid. Second, a clear path is known to exist (at least the first time a grid is built). Third, obstacles will eventually move out of the way (or be passed). Fourth, the target position is re-computed (by interpolation) each time a grid is built.

The immediate planning algorithms first select a target cell and then select a speed as a function of the target cell's value. If the value is negative a clear path to the target is presumed to exist and so the full

specified speed is used by the vehicle. If the cell has a positive value, the vehicle's speed is reduced.

3.4 Avoiding Collisions

To avoid colliding with a moving object, two options exist: change speed or direction. In real life a speed change could involve an increase in speed, but for our work we use only deceleration from the vehicle's desired speed.

The DOA algorithms look at cells' values both for direction (smaller values generally indicate more desirable paths) and speed (smaller values indicate higher speeds can safely be used). After the direction is selected, the relative elevation of the cell being entered is used to select a speed. If the cell's value is less than zero, a clear path to the target is at hand and the vehicle moves at full speed. A relatively large elevation indicates the cell is near obstacles and lower speeds are selected.

3.5 The DOA Testbed

In order to better understand the DOA Model prior to implementing it within the IST CGF Testbed, a stand alone DOA Testbed was created. The DOA Testbed allows experimentation with DOA approaches without the complexities of terrain navigation and vehicle dynamics within a DIS environment. Although the DOA analysis was moved to the IST CGF Testbed so simulated vehicle behavior could be directly examined, the fundamental work and algorithm selection was done in this stand alone DOA Testbed.

4. IMPLEMENTING THE DOA MODEL IN THE IST CGF TESTBED

Behavior control within the CGF Testbed is implemented through a code structuring technique based on Finite State Machines (FSMs)(Smith, 1992). An FSM manages task resources and scheduling in a manner similar to that of a process in a multitasking operating system. It isolates and protects its state information much as an object does in an object oriented programming environment.

An FSM encoding the DOA Model (the DOA FSM) was added to the CGF Testbed. This FSM schedules itself based on the speed of the vehicle and awakens the ManueverToPoint FSM (see below) to make suggested changes in vehicle speed and direction.

4.1 The ManeuverToPoint FSM

Within the CGF Testbed, routes are piecewise linear curves represented by a list of points. These routes are generated by a vehicle level route planner which plans route around static obstacles. Route following causes entities to maneuver towards the first route point on its route point list. As each route point is reached, it is removed from the route point list, causing the vehicle to move towards the next point on the route.

Route following is implemented in the ManeuverToPoint FSM. The states within the ManeuverToPoint FSM set requested values (such as, requested speed, requested turn) which are used by the entity's dynamics process. The ManeuverToPoint FSM maneuvers the entity so that it passes near each route point and comes to a stop near the last route point on the route point list.

4.2 Maneuver control with the DOA Model

The DOA FSM is started for a vehicle when the vehicle is beginning to move along a route and terminated when the vehicle reaches its destination.

The DOA FSM performs a "snapshot" analysis of the local situation, makes recommendations for speed and heading, and schedules itself to be repeated in the near future. The scheduled time is proportional to the speed of the vehicle so that the vehicle moves less than the width of a grid cell between DOA analyses.

4.3 DOA Algorithm

The DOA algorithm is:

1. Fill the DOA grid. Grid cells are marked with obstacles (moving vehicles), barriers (stationary vehicles, static objects), the interpolated target location, the vehicle's location, and information about the terrain surrounding the vehicle.
2. Propagate cell values within the grid, as described in Section 3.2.1.
3. Apply the algorithm specified in the configuration file. A suggested speed and heading are produced.
4. Change the vehicle's speed and heading:

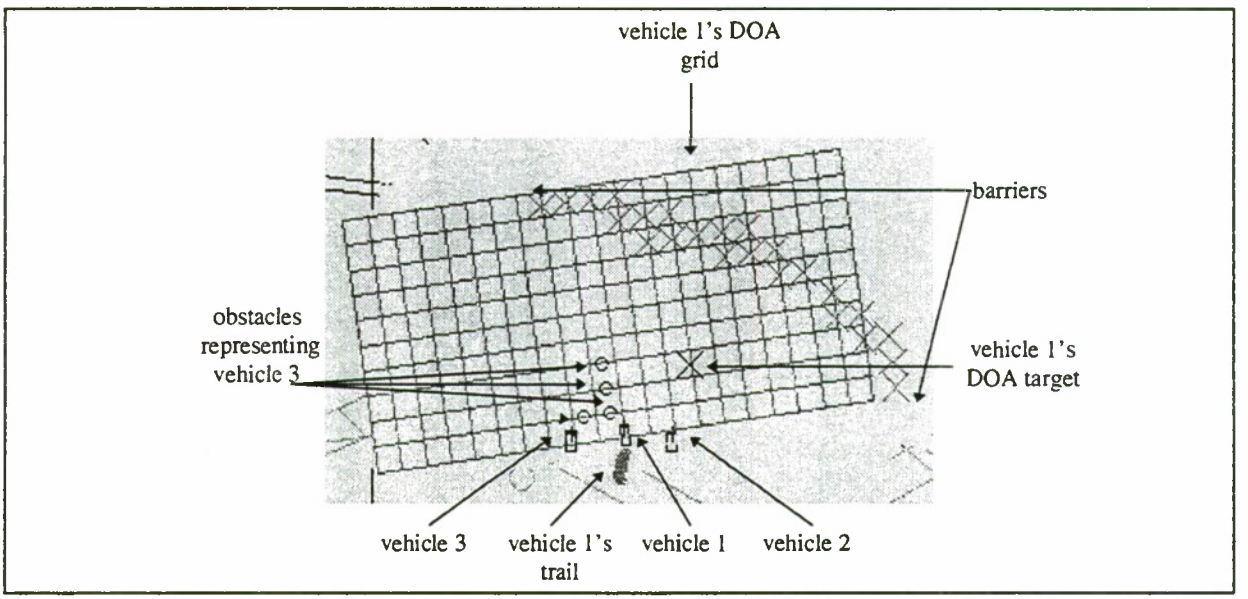


Figure 4: DOA Grid

a. If the nearest vehicle is less than 20 meters away, slow to a stop (i.e. halt).
b. If the nearest vehicle is between 20 and 30 meters away and to the right, slow to a stop.
c. If the vehicle has been halted and should remain halted, backup.
d. If the vehicle has been halted and doesn't need to remain halted, resume movement.
e. If the suggested speed is less than a threshold, maneuver along the suggested heading with the suggested speed.
f. Otherwise, use the suggested speed heading directly at the target.

Steps 1-3 are straight forward. Step 4 makes the final maneuvering decisions. Steps 4a-d encode rules for resolving imminent collision situations:

a) Rule 4a causes the vehicle to begin abrupt braking if a collision seems imminent.
b) Rule 4b encodes a "yield to the right" rule of the road.
c) Rule 4c allows the vehicle to backup out of deadlock situations.
d) Rule 4d permits the vehicle to resume movement after halting and backing up.

Rules 4e and 4f accept the DOA Model suggestions. 4e causes the vehicle to follow the speed and heading suggestions while 4f accepts only the speed suggestion.

In summary, the DOA algorithm causes vehicles to avoid collisions by:

a) Slowing first (rule 4f),
b) Then slowing and steering away from obstacles (rule 4e),
c) Then stopping (rules 4a and 4b), and finally
d) Backing up (rule 4c).

Of course only the maneuvering that is necessary to avoid collisions is performed; e.g. if slowing resolves the problem then steering, halting, and backing up are not utilized.

4.4 The DOA grid

Figure 4 depicts three M1s showing only Vehicle 1's DOA grid. The DOA grid is 100m by 200m with 10m by 10m grid cells. This allows Vehicle 1 to do perform localized DOA 100 meters to its front. Vehicle 1 is using DOA to maneuver around obstacles (Vehicle 2 and Vehicle 3) and a barrier (a river).

Notice that Vehicle 3's projected position creates obstacles in front of Vehicle 1. The DOA Model has suggested a target location for Vehicle 1 to the right so as to avoid Vehicle 3. Vehicle 2 is behind and to the right of Vehicle 1 and does not influence Vehicle 1's DOA grid.

The DOA grid is a rectangular array of cells, each contains:

1. Contents of a cell:
   - EMPTY: If it contains only open terrain,
   - DOA_VEHICLE: If it contains the vehicle for which the DOA analysis is being performed,

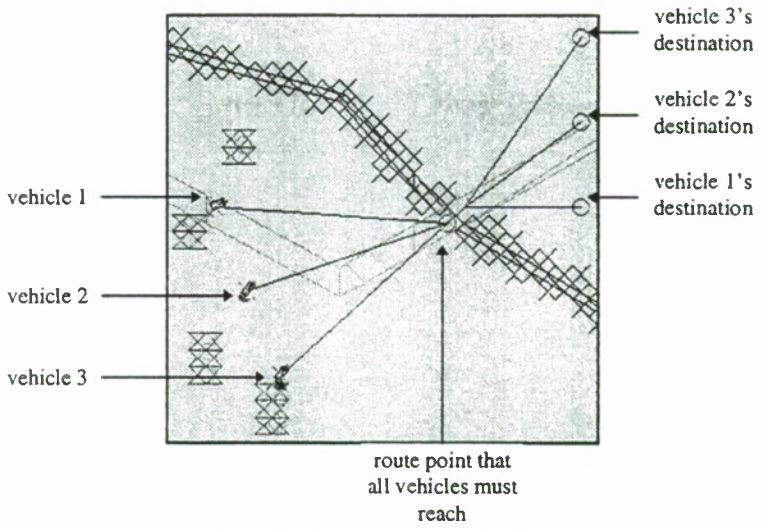


Figure 5: Competition for the Bridge

- TARGET: If it contains the target location to which the DOA_VEHICLE is trying to reach,
- OBSTACLE: If it contains a moving vehicle or a moving vehicle's projected position, or
- BARRIER: If it contains a stationary object or barrier; e.g. a river.

2. Value of the cell:
   - The propagation of cell values within the grid if it is an EMPTY cell, or
   - The fixed value used for DOA_VEHICLE, TARGET, OBSTACLE or BARRIER.

The DOA grid is placed in front of the vehicle oriented along the vehicle's heading. The vehicle is located at the "rear" center of the DOA grid. This arrangement allows the vehicle to detect and avoid obstacles and barriers on its forward path. Passed obstacles and barriers drop off the vehicle's DOA grid, and stop influencing its maneuvering.

5. DOA MODEL RESULTS IN THE IST CGF TESTBED

Experiments with the DOA Testbed revealed that analyzing the DOA grid with the A* algorithm (step 3 in Section 4.3) gave the most stable and realistic results. Five different scenarios using A* based DOA analysis were developed to test the DOA Model:

1. X-scenario (two vehicles moving in a 315 and 225 degree collision course),
2. Head On Collision scenario (two vehicles moving in a head on collision course),
3. Right Angle Collision scenario (two vehicles moving in a 0 and 270 degree collision course),
4. Competition for the Bridge scenario (three vehicles competing to cross a bridge), and
5. Head On Collision On the Bridge scenario (two vehicles moving in a head on collision course on a bridge).

The DOA Model produced realistic driving behavior in all five scenarios. Space limitations allow only one scenario (4. Competition for the Bridge) to be discussed in detail.

5.1 Scenario 4: Competition for the Bridge.

In contrast to the other scenarios, this scenario involves three vehicles moving in the same direction. The complication is that they must all cross a narrow bridge. The scenario is arranged so that the vehicles would arrive at the bridge simultaneously without DOA. Figure 5 shows the vehicles in their starting locations and their routes. The routes converge on the bridge

Figure 6 shows the vehicles at their destinations. For clarity, the vehicles and their "trails" are shown in the Figures; the DOA grids are not shown. The density of trail marks indicates the vehicle's speed; the denser the trail the slower the vehicle was moving.

In this scenario, the three vehicles were ordered to proceed at normal speed with DOA active. All maneuvering and speed changes were the results of the DOA Model.

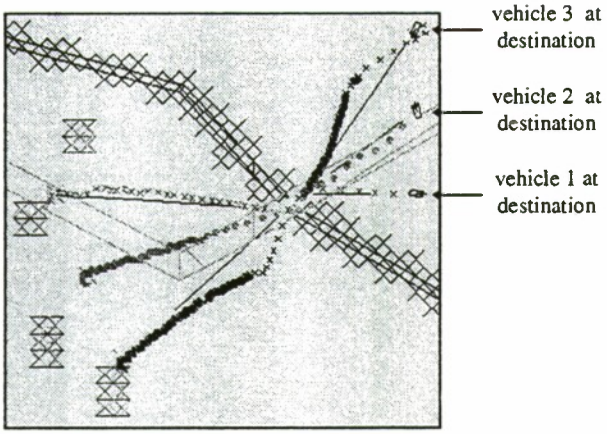


Figure 6: Competition for the Bridge

The maneuvers made by each vehicle in this scenario were:

Vehicle 1:

1. Started to move slowly towards its first route point.
2. Broke loose from the other vehicles influence and accelerated.
3. Crossed the bridge at normal speed.
4. Reached its destination.

Vehicle 2:

1. Started to move slowly towards its first route point.
2. Continued to move slowly to avoid Vehicles 1 and 3.
3. Moved a little bit ahead of vehicle 3.
4. Crossed the bridge at normal speed.
5. Approached destination.
6. Reached its destination.

Vehicle 3:

1. Started to move slowly towards its first route point.
2. Continued to move slowly to avoid Vehicles 2 and 3.
3. Decelerated to avoid Vehicle 2.
4. Steered to its right to avoid Vehicle 2.
5. Crossed the bridge at normal speed.
6. Decelerated after crossing bridge to avoid Vehicle 2.
7. Accelerated after Vehicle 2 stopped.

In summary, the vehicles, through a combination of deceleration and minimal steering, passed over the bridge without collision. This scenario demonstrates that the DOA Model resolves conflicts at chokepoints without a complex set of "rules of right-of-way".

6. CONCLUSIONS AND FUTURE WORK

IST has developed a novel approach to attack the DIS dynamic obstacle avoidance (DOA) problem by combining two disparate motion planning approaches: potential field and regular grid analysis. This approach is rooted in neural net fundamentals and the underlying design allows various techniques to be brought to bear on the avoidance problem. To allow focused study of the DOA problem, IST developed a stand alone DOA Testbed. On this foundation, IST has implemented and evaluated many techniques which would seem inapplicable using other approaches (from a simple "best guess" method to spline fits). To test the validity and applicability of these results, IST implemented the more successful DOA algorithms within its CGF Testbed and studied their results within a DIS environment. In particular, the A* based DOA Model shows excellent moving obstacle avoidance while maintaining reasonably close adherence to previously created piecewise linear routes.

There are several opportunities for further work in the area of Dynamic Obstacle Avoidance. Among them are the real time coordination of route following, station keeping within formation, and dynamic obstacle avoidance. This work has focused on dynamic obstacle avoidance within the context of following lengthy routes generated by route planners that ignore dynamic (moving) obstacles. The coordination of dynamic obstacle avoidance and station keeping within a formation is an interesting area for further research.

7. ACKNOWLEDGMENT

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

8. REFERENCES

Charniak, Eugene and McDermott, Drew (1987). *Introduction to Artificial Intelligence*, Addison-Wesley.

Cortes-Rello, E. and Golshani, F. (1990). "Dynamic Route Planning", *Proceedins of the Fifth Rocky Mountain Conference on Artificial Intelligence: Pragmatics in Artificial Intelligence*, Las Cruces, NM, pp 87-92.

Craft, M. A., Cisneros, J. E., and Karr, C. R. "Dynamic Obstacle Avoidance", *Technical Report*

IST-CR-94-41, Institute for Simulation and Training, University of Central Florida, 54 pages.

Karr, C. R., Craft, M. A., and Cisneros, J. E. (1995) "Dynamic Obstacle Avoidance" *SPIE Proceedings CR58*, Orlando, Florida, April 17-21 1995.

Glasius, R., Komoda, A., and Gielen, S. (1994). *"Neural Network Dynamics for Path Planning and Obstacle Avoidance",* Neural Networks, March 1994 (to appear).

Hertz, John, Krogh, Anders, and Palmer, Richard G (1992). *Introduction To The Theory Of Neural Computation*, Addison-Wesley.

Hwang, Y. K. and Ahuja, N. (1992). "Gross Motion Planning--A Survey", *ACM Computing Surveys,* Vol. 24, No. 3, pp.219-291.

Mitchell, J. S. B. (1988). "An Algorithmic Approach to Some Problems in Terrain Navigation", *Artificial Intelligence,* Vol. 37, pp. 171-201.

NASA (1991), "Potential-Field Scheme for Avoidance of Obstacles by a Robot", *NASA Tech Brief KSC-11491*, John F. Kennedy Space Center, Fl., pp. 1-43.

Roos, T. and Noltemeier, H. (1991). "Dynamic Voronoi diagrams in Motion Planning", *Proceedings of the International Workshop on Computational Geometry,* Bern Switzerland, March 1991, pp 228-236.

Smith, J. (1994), "Near-term Movement Control in ModSAF", *Proceeding of the 4th Computer Generated Forces and Behavioral Representation Conference*, Orlando Florida, May 4-6 1994, pg. 249-260.

Smith, S. H., Karr, C. R., Petty, M. D., Franceschini R. W., and Watkins, J. E. (1992). "The IST Computer Generated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, University of Central Florida.

Thorpe, C. E. (1984). "Path Relaxation: Path Planning for a Mobile Robot", *CMU-RI-TR-84-5*, Carnegie-Mellon University The Robotics Institute Technical Report, April 1984.

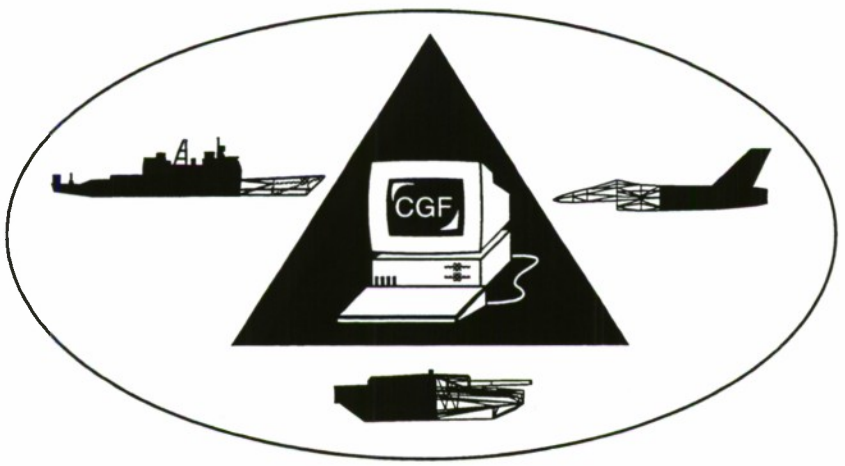
Winston, Henry Patrick (1992). *Artificial Intelligence*, Third Edition, Addision-Wesley.

9. AUTHORS' BIOGRAPHIES

Clark R. Karr is the Computer Generated Forces Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Michael A. Craft is a Research Associate working for the Intelligent Simulated Forces project at the Institute for Simulation and Training. He has a Master of Science degree in Mathematics and a Master of Science in Computer Science. His research interests are in the areas of Software Engineering, Real Time Systems, Simulation, and Network Protocols. Mr. Craft has a broad and diverse background ranging from Office Automation to Missile Tracking Systems.

Jaime E. Cisneros is a Research Associate in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Cisneros has a Masters of Science degree in Computer Science, and is currently working on a Masters of Electrical Engineering. His research interests are in the areas of Natural Language Understanding, Machine Learning, and Computer Generated Forces.


CGF

Path Planning with Terrain Utilization in ModSAF

Bruce Hoff[1], Michael D. Howard, and David Y. Tseng
Information Sciences Laboratory
Hughes Research Laboratories
3011 Malibu Canyon Rd., Malibu CA 90265

1. Abstract

The development of infantry operations in Marine Corps SAF requires the use of small scale terrain features for cover and concealment. A useful method of incorporating such features in path planning involves the assignment of cost factors to each feature; e.g. low costs assigned to highly protected terrain, and high costs to exposed areas. The appropriate representation of the terrain plays an important role in selecting the best path from a start point to a destination and in maximizing computational efficiency. We compare the suitability of grid-base (GB) and weighted-region (WR) based terrain representations for planning.

We have applied the weighted-regions approach to the path planning needs of Marine Corps SAF in ModSAF [presented in a companion paper by Howard, et al.]. Because terrain features in ModSAF are represented as disjoint polygons superimposed on a uniform background, while the weighted-regions representation requires a "tiling" of the plane made up of weighted contiguous polygons, an interface was required to bridge these two representations. We will describe this interface and discuss the application of the resulting system for the path planning needs of infantry movements. Finally, future research issues relating to geometrical terrain analysis and path planning are discussed.

2. Introduction

The development of military simulations has focused on *Semi*-Automated Forces (SAF). This reflects the need for entities which, while not having complete autonomy, exhibit enough intelligence to fill in the behavioral details when assigned a task by a human operator. While much of the Computer Generated Forces (CGF) effort has focused on armored vehicles and aircraft, Marine Corps SAF requirements include, as a major part, the need for SAF entities which are simulated infantry, or "Individual Combatants" (ICs). Actual infantry units, when moving under threat of enemy fire, move both quickly and with maximum use of cover and concealment. Therefore, an IC-SAF unit ought to be capable of making a judicious choice of route, when tasked to move to a specific battlefield destination, thus freeing the workstation operator from having to study the terrain and make the route choice.

For both the Marine Corps LeatherNet project and the IC-SAF portion of STOW97, we were given the mission of building IC capabilities into the latest version of ModSAF (version 1.2, when we began in mid-1994). One of the basic "behaviors" to be developed was the ability to move using the cover and concealment afforded by the surrounding terrain. ModSAF operates in the context of a terrain database, which includes an elevation map and terrain features such as obstacles. ModSAF obstacles include buildings, lakes, rivers, and tree canopies (the latter being more of an obstacle to armored vehicles than to ICs). These features are internally represented as planar polygons, stored by their vertices, and annotated with other descriptive information (Loral 1994).

ModSAF has the capability of generating obstacle free paths: Given a destination for a vehicle situated in the terrain, ModSAF will generate a path for the vehicle which skirts the intervening obstacles. In the "spirit of SAF", an operator can issue a Move task to a vehicle, which will then find its way to its goal, avoiding obstacles in a natural way (Loral 1994). However, ModSAF lacked the representation of "desirable" terrain regions, i.e. regions, such as covered-or-concealed corridors, which are somehow advantageous to a vehicle. It therefore also lacked the ability to plan paths which use such desirable regions. (Version 1.4 of ModSAF begins to provide such a capability.) We needed to model the ability of human ICs to choose paths that take advantage of cover and concealment, when moving to a destination.

The remainder of this paper describes the modeling of path choice, by turning to a family of terrain feature representations and their associated path-planning algorithms. We explain our specific representation and algorithm selections and the interface between the chosen path-planning algorithm and the ModSAF

[1]To whom correspondence should be addressed. Internet: hoff@isl.hrl.hac.com

system. We show the results of the enhanced system, and discuss prospects for future research.

3. The "Shortest-Path" Paradigm

The problem of choosing a path through the terrain which maximizes the use of cover and concealment can be couched in terms of a "shortest-path" problem (Mitchell 1988). In the "shortest-path" paradigm, sections of terrain are weighted with non-negative costs. The cost for a path is computed by summing the costs incurred in each section. By assigning low costs to highly concealed or covered sections of terrain, and high costs for exposed sections, we convert the problem of finding the most covered-and-concealed path to the problem of finding the path that has the lowest cost (also called the "shortest path" from the analogy of cost to length). The cost values may be computed based on a number of algorithms, including line-of-sight from a known enemy location. Speed of movement through the terrain may be factored in, so that exposure in a "slow-go" region is more costly that exposure in a "go" region. Impassable regions (obstacles) are assigned very high, or even infinite, weights so that there is no choice of start and end points for which the shortest path passes through such a region. Clearly there is a need for a wide spectrum of cost values, so that graded levels of region desirability may be defined.

There is a choice in the representation of the weighted terrain. In a grid-based (GB) representation, the plane is divided into small, regular regions, e.g. equal sized squares. The center of each region is connected to that of its neighbors with weighted edges. The allowed paths travel along these connecting edges, through the regions, and are assigned costs which are the sum of the weights of the edges traversed. The path choices are discrete, bound to the shape of the grid. (This constraint is called *digitization bias* by Mitchell, 1988.) There are a finite number of simple (non-self-intersecting) paths between any two points.

In a weighted regions (WR) representation the plane is "tessellated" by polygonal subdivisions, each weighted with a non-negative cost-per-unit-length. The cost for a path within each section is the distance the path takes through each section times the weight assigned to that section. The total cost for the path is the sum of the costs through each section. The space of paths is continuous: Any locus of points contained in the plane, connecting a start and end point, is valid in this representation, and there are an infinite number of simple paths connecting any two points. Intuitively, the WR representation is more natural yet less computationally tractable than the GB representation.

An advantage of the GB representation is that it affords well understood solutions to the problem of finding the shortest-path, e.g. Dijkstra's algorithm (Dijkstra 1959) which has worst case running time of O(n log n) where n is the number of vertices in the graph. The solution to the WR problem is less well known, but a polynomial time algorithm for finding near optimal paths has been developed (Mitchell and Papadimitriou 1991) and will be discussed in Section 4.

A strong influence on our choice of representation for IC-SAF is the problem the GB approach has with capturing small terrain features: The grid spacing must be as small as the smallest feature to be captured. A representative terrain database (discussed in Section 6) has size of about 1.5 km x 1.5 km. Assume that terrain features important to IC path planning are on the order of the size of ICs. In ModSAF's physdb.rdr file, the width of an IC is specified as 0.5 m. The number of graph vertices necessary to allow the representation of features of that magnitude in a uniform rectangular grid is then $(1.5 \text{ km} / 0.5 \text{ m})^2$, or about 10^7 vertices. In contrast, the WR approach allows us to specify polygonal regions of any size, so that arbitrarily small features may be represented by small polygons at the same time that a few large polygons describe featureless sections of terrain. The total number of WR graph vertices is based only on the number of vertices in the polygonal features, and is independent of the dimensions of the terrain database. In the application described in Section 6, the planning graph, which is based on manually entered terrain features, has less than 30 vertices. In a planned extension to the algorithm (discussed in Section 7) which would generate the features automatically from ModSAF's microterrain, the graph would still only be about 4×10^3 vertices (and about 10^4 edges), several orders of magnitude smaller than the corresponding grid.

Because of the polygonal feature representation existing in ModSAF, the lack of digitization bias and relatively small graph size in WR, and the availability of a WR-based path planner (presented below), we chose to adopt the Weighted Regions approach for IC-SAF path planning. (We note that as of this writing, the latest version of ModSAF has gained a grid-based cover-and-concealment route planner, and acknowledge that such a choice of representation may be suitable for armored vehicles, concerned with larger scale terrain features.)

4. The Weighted Regions Algorithm

The weighted regions algorithm is quite complex, and is described in detail by Mitchell and Papadimitriou (1991). Here we give the flavor of the algorithm, by describing the "Continuous Dijkstra" approach which it employs, the local optimality properties of shortest paths, and the ray tracing algorithm used to produce paths.

In a grid based representation, the shortest path can be readily solved by a straightforward technique by Dijkstra (1959): From the start point, propagate through the grid step-by-step such that at any step a collection of grid points is maintained all of which are the same distance from the start, when connected to the start by their shortest path. This collection of grid nodes can be thought of as a "wave-front" of constant cost. When this front meets the end point, the search is complete. In the Continuous Dijkstra paradigm, a continuous analog of this wavefront is computed. The WR algorithm generates this wavefront along a discrete set of rays, projected outward from the start point in all directions. Each ray obeys local optimality criteria, and so provides a candidate shortest-path. The idea is to trap the goal point within a two-dimensional "cone" whose boundaries are two projected rays.

The local optimality criteria obeyed by the projected rays, are that 1) rays travel in straight lines through regions of constant weight (as the regions in the WR representation), 2) rays refract across boundaries between regions of different weights according Snell's Law (which is a commonly known property of optics), and 3) rays don't strike boundaries at angles greater than the critical angle defined by Snell's Law. Local criteria for projecting paths break down when the path either 1) strikes a region vertex, or 2) strikes a region edge at a critical angle, refracting along the edge, and traveling on that edge for a distance unpredictable from local information. These breakdowns in the determination of the optimal path from local criteria cause the algorithm to revert to an exploration of alternative paths. However, this search is reduced by the important theoretical finding by Mitchell and Papadimitriou (1991) that "between any critical point of exit and the next critical point of entry, there must be a vertex." This property places a tight rein on the otherwise arbitrary length of the critical reflection section of an optimal path.

The algorithm is ε-optimal, meaning that the derived path is no longer than $(1+\varepsilon)$ times the optimal path length, for some small, preset ε. This tolerance, ε, is related to the closeness of neighboring rays between which the goal point is trapped: The more rays utilized, the closer the rays come to the goal point, and the closer the path is to being optimal. The use of an ε-optimal algorithm, rather than an optimal algorithm, results in an algorithm which has polynomial time complexity, rather than exponential time complexity. The complexity of the algorithm is O(ES) where E is a number of "critical events" in the algorithm (occasions when a ray strikes an edge or vertex), and is at most $O(n^4)$ (where n is the number of vertices). S is the complexity of the processing done for each event, and is at worst $O(n^4)$ also. Thus the entire algorithm then has worst case complexity $O(n^8)$, but Mitchell and Papadimitriou have found that E and S may be much smaller in practice, and thus the algorithm may often run much faster than this worst case complexity.

The following section discusses the interface of ModSAF's feature representation to the WR path planner, after which we present simulation results for the integrated system.

5. Path Planner to ModSAF Interface

In adding terrain utilization to ModSAF's path planning, the programming task was to interface ModSAF's feature representation to the path planner's weighted region representation. ModSAF represents obstacles (lakes, buildings, etc.) as non-overlapping polygons lying on a background plane. To be consistent, we represent desirable terrain regions in the same way. The concept is illustrated in the first panel of Figure 1, where the light and dark polygons are features, lying on a background plane of uniform cost. The existing obstacle polygons represented in ModSAF are copied into a data structure called a "movemap" which is created for each vehicle. We add to this data structure a list of cover-and-concealment features, which are read in from a file.

The WR path planner places several requirements on the format of its data. The map must be composed of polygons, each assigned a positive numerical weight. Further, each polygonal region must be convex, meaning that no interior angle may exceed 180°. Lastly, the polygons must fit together like a puzzle, creating a contiguous tiling of the plane. We use three weight values for the regions: a "medium" one for the background plane, a "high" one for obstacles, and a "low" one for desirable features, as shown in the first panel of Figure 1. This diagram shows two violations of these WR requirements: First the light region on the left is not convex, having an interior angle greater than 180°. Second, the two polygonal features alone don't create a contiguous tiling of the plane. We can solve this second problem by viewing the background as a weighted region. Now we do

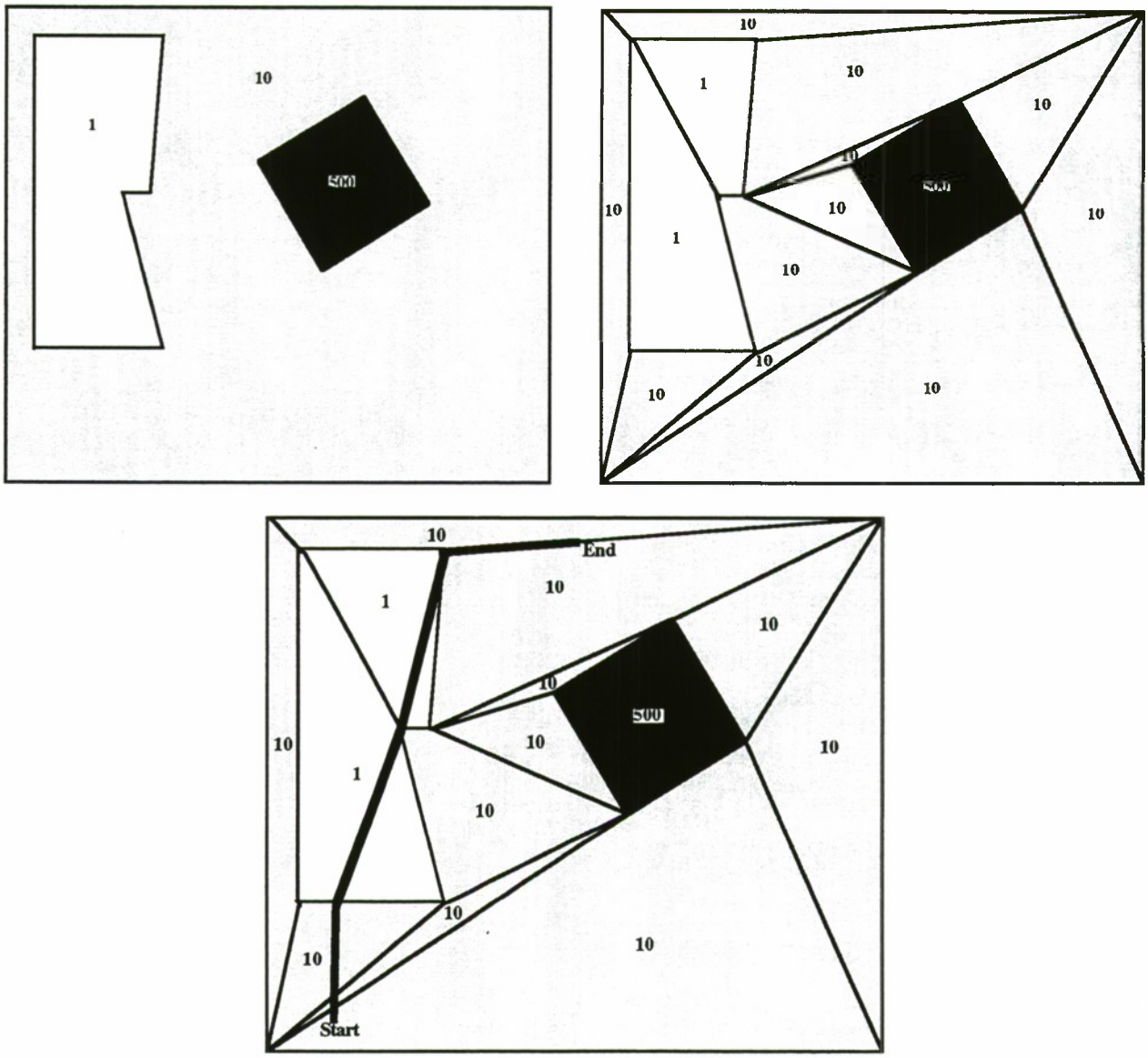


Figure 1: Top left: In the ModSAF style of terrain feature representation, disjoint polygons lie in a background plane. We can assign a "cost" to each polygon, and to the background, to represent the penalty for planning a path through that region. By giving the region on the left a cost *less* than that of the background, we specify that movement through that region is more desirable than movement through open space. We give obstacles weights much higher than the background, so that low cost paths avoid them. Top right: Since the Weighted Regions algorithm requires a contiguous tiling of the plane by simple, convex regions, we subdivide non-convex regions (e.g. the region on the left) and the background plane, by adding an edge whenever we find an angle greater than 180° at a polygon vertex. Bottom: A minimum-cost path from "Start" to "End" is shown. Note the deviation through the low cost region and the locally optimal "Snell's Law" refraction when crossing the boundary between the background and the low-cost region.

have the requisite contiguous tiling, but have introduced another illegal polygon, since this background plane has many concave angles. The second panel of Figure 1 shows the solution to the concavity problem: For each angle greater than 180°, there is another vertex in the graph such that an edge can be added from that vertex to the vertex possessing the concave angle, splitting the concave angle in two, and splitting some weighted region in two also. The two new regions are each weighted as was their parent region. Edges can always be added, until no concave angles remain. (The extreme case would be to keep dividing regions until all the regions were triangles, which are necessarily convex.) The result is the required tessellation of the plane with convex polygons.

The complete translation algorithm can be summarized as follows:

1) From the input polygon list, build a list of vertices.
2) Add the vertices of a bounding box (or of any convex polygon surrounding the given regions).
3) Build a visibility map for each vertex (a list of vertices reachable from that vertex without crossing an existing edge), and note whether a connecting edge to a visible vertex exists already.
4) For each concave angle (except the exterior angles of the bounding box), add edges from the vertex of the angle to another vertex which is in its visibility list and which is "inside" the concave angle. More than one edge per concave angle may be added.
5) Pass the resulting convex, weighted tiling to the WR planner.

To apply path planning to IC movement, we intercept the IC unit's planned route, in the appropriate ModSAF move task: VMove, UTraveling, or UCMarch. We take each waypoint in the planned route as an unchangeable constraint, and optimize the sub-paths in between the waypoints: We pass to the WR path planner each route point and the one which follows it. The path planner returns the minimum-cost path connecting the two points. The third panel of Figure 1 illustrates the result of optimizing a path: The two point route from "Start" to "End" is transformed by the planner into an optimized route containing five points. After repeating this optimization for all route segments, we assemble the returned paths into the unit's complete route, and return it to ModSAF.

The following section describes sample results of the integrated ModSAF/WR system.

6. Application to the "LeatherNet" Project

A goal of the Marine Corps LeatherNet project is the creation of a training and simulation environment for Marine exercises. One of the training "ranges", Range 400, has been digitized at the one meter level, for the purpose of computer simulation. A ModSAF "CTDB" format database with imbedded microterrain based on this data has been created. The exercise carried out on Range 400 is a company-size deliberate attack. When the Marines move up the range toward their objectives, they choose concealed routes. While no tree canopies or buildings exist on this range, there are a number of deep wadis (dry stream beds) that provide concealment during the advance. Accurate simulation of Range 400 exercises therefore requires representation and usage of these wadis in route planning.

The top of Figure 2 shows a plan view of Range 400, as displayed by ModSAF. The terrain database provides the topography and roads. The borders of concealed regions, provided by wadis, were manually extracted from an elevation map, and listed in a file in terms of the vertices of the resulting polygons. These polygons were automatically added to the movemap data structures of the ICs. The polygons are shown as shaded regions in Figure 2. Also shown in this Figure are a Marine fire team, situated at the southern end of the range, and a goal position at the northern end.

When the first movement command is issued to the unit in ModSAF, the polygonal features are processed as described in Section 5. They are subdivided into convex regions and assigned the appropriate weight. (The process is illustrated in Figure 1.) The uniform background is also subdivided into convex regions. The resulting map is then passed to the WR path planner. The path generated by the planner is shown at the bottom of Figure 2. The length of the path outside the wadis has been minimized, thus maximizing the use of cover and concealment.

Figure 3 shows a situation where the minimum-cost path doesn't use the wadis at all. The algorithm has discovered that deviation of the straight-line path (connecting the unit to its goal) to use the available concealment would add more exposure (for the sections of the route that carry the unit to and from the concealed regions) then it would remove by utilizing those regions. Figure 4 gives an example (in a different terrain database) of the algorithm simultaneously making use of desirable regions and deviating around obstacles. The unit detours around a lake to reach its destination, and also uses two of three available low-cost regions in its minimal path.

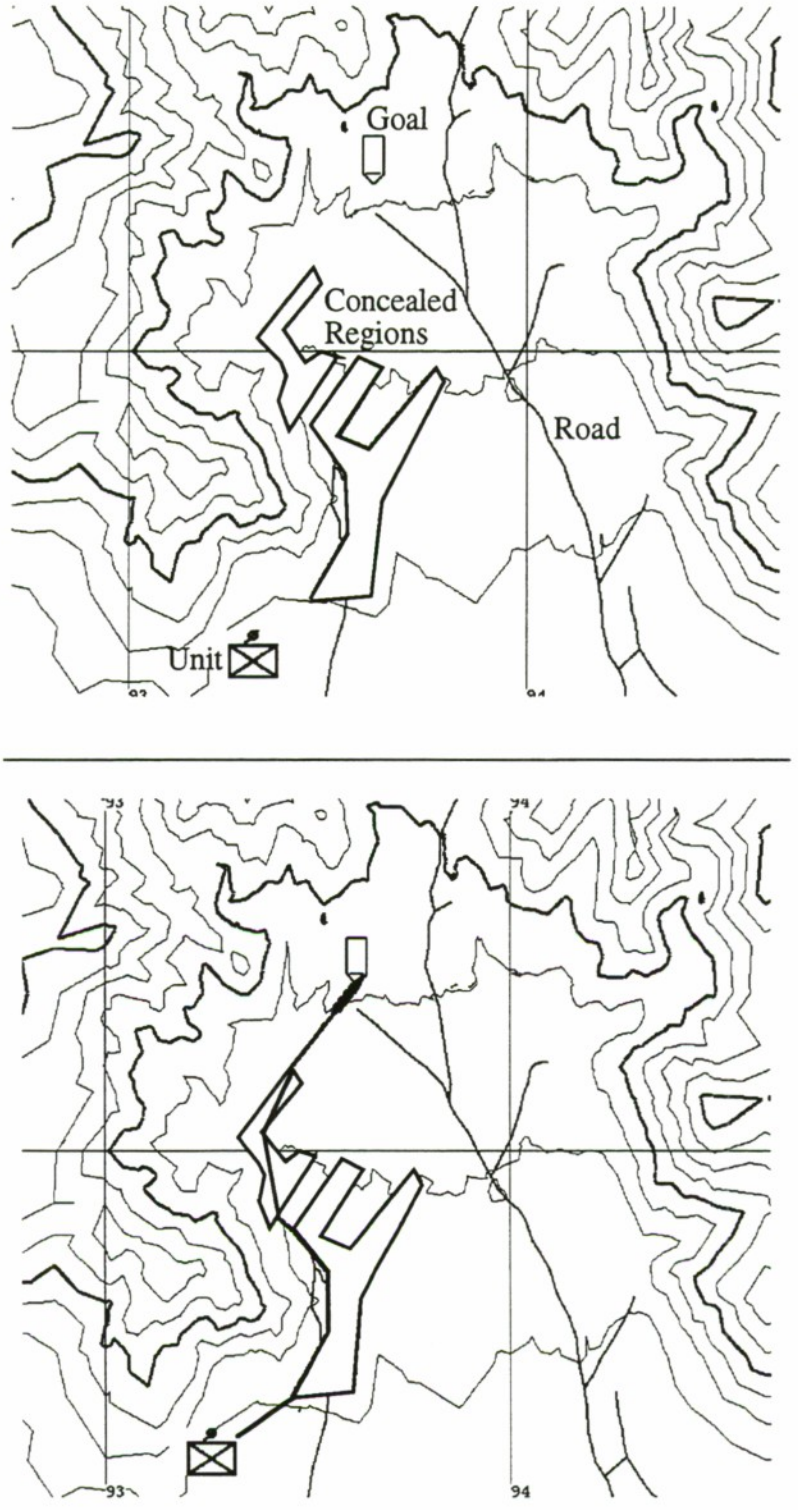


Figure 2: Top: Plan view of sample terrain database. Shown are topographical lines, roads, concealed regions (shaded polygons), a unit icon (box near bottom), and the unit's assigned goal (near top). The path planning algorithm considers only the concealed regions, obstacles, and the start and goal points for the unit. Bottom: Results of planning the "shortest" (i.e. minimum-cost) path from unit's position to its goal. The shown path is the shortest in terms of the amount of exposed terrain (outside concealed regions) used.

7. Future Work

In the application discussed in Section 6 the low-cost terrain regions were added manually. The primary extension to this work is to derive the low-cost regions from the terrain database. ModSAF represents small scale terrain elevation by patches of microterrain in the form of contiguous, non-coplanar triangles, called triangulated irregular networks (TINs). The main 1.5 km x 1.5 km section of the ModSAF Range-400 database, for example, contains a TIN which has about $4x10^3$ vertices (triangle corners). Visibility from a given enemy position could be calculated for each vertex (using ModSAF's point-to-point visibility algorithm), and the triangles would be weighted appropriately. The weighted triangles would then provide the weighted regions for the WR path planner. Returning to the discussion in Section 3 on the choice between WR and GB representations, we see that for such a vertex-visibility approach, the WR representation would have far fewer vertices than the 10^7 vertices of the alternative grid.

In using the TIN triangles as weighted regions, one approach is to project the triangles into the horizontal plane, creating the familiar planar tiling. An alternative is to plan on the non-planar TIN map itself. Mitchell (1988) has examined a form of this problem called the Discrete Geodesic Problem, in which the regions have uniform weight. A related issue is directionality of region weight. For instance, we might want to assign a high cost to a region of steep terrain, but only if the path ascends that region. A high cost should not be assigned if the path merely traverses the region at constant elevation. This problem has been examined by Rowe and Ross (1990).

In the applications discussed above, only three region weights were used (to differentiate desirable regions, nominal regions, and obstacles). In general, however, the WR approach allows an entire spectrum of weights. Applications of this capability include representation of graded levels of visibility, combinations of visibility with other factors (e.g. maximum speed for each region), and more complex models of danger to the traveler. Other factors can also be entered by using a *bi-criteria* planning paradigm, in which we minimize the path *cost*, subject to some *constraint*. An example is movement with maximum cover and concealment (the cost) subject to the constraint of arriving within a certain time limit.

The path planning approach of this paper is based on a complete map, giving unrealistic terrain omniscience to the traveling unit. A more realistic approach would incorporate a "visibility horizon", which would only provide terrain knowledge for a certain radius around the moving unit. This problem has been studied by others (e.g. Brock et al., 1992).

We have only discussed polygonal features, but we might also add such linear features as roads and rivers, which could be handled by the WR path planner much as it handles edges of polygons.

We have assumed a static planning map, but in real life movement takes place within a constantly changing environment. A complex problem is that of planning routes that anticipate enemy movements, represented as changing levels of exposure for regions over time. (A hidden region may become exposed to the enemy as he moves toward it.)

Alternatively, we may remain within the static map paradigm, replanning when changes to the map occur. The issue is then how to take advantage of the computation already completed, so that the revision is done faster than if the plan were computed from scratch.

8. Acknowledgments

The research reported in this paper was supported in part by ARPA contract DAAE07-92-C-R007, contracted through US Army TACOM. We gratefully acknowledge the continued guidance and support of LCDR Peggy Feldmann and LT Jeff Clarkson in the performance of this program. Our colleagues Prof. Joseph Mitchell and Christian Mata of State University of New York at Stony Brook collaborated closely with us in the planning aspects of the problem, and their weighted regions algorithm plays a crucial role in our approach. Dr. Jimmy Krozel, of Hughes Research Laboratories, provided useful, theoretical discussion, and relevant references, on path planning alternatives. The support and encouragement from Scott Harmon is very much appreciated.

9. References

Brock, D.L., Montana, D.J., Ceranowicz, A.Z. (1992) "Coordination and Control of Multiple Autonomous Vehicles", Proceedings of the 1992 IEEE International Conference on Robotics and Automation, 2725-2730.

Dijkstra, E.W. (1959) "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, 1: 269-271.

Loral (1994) ModSAF Software Architecture Design and Overview Document, Loral Advanced

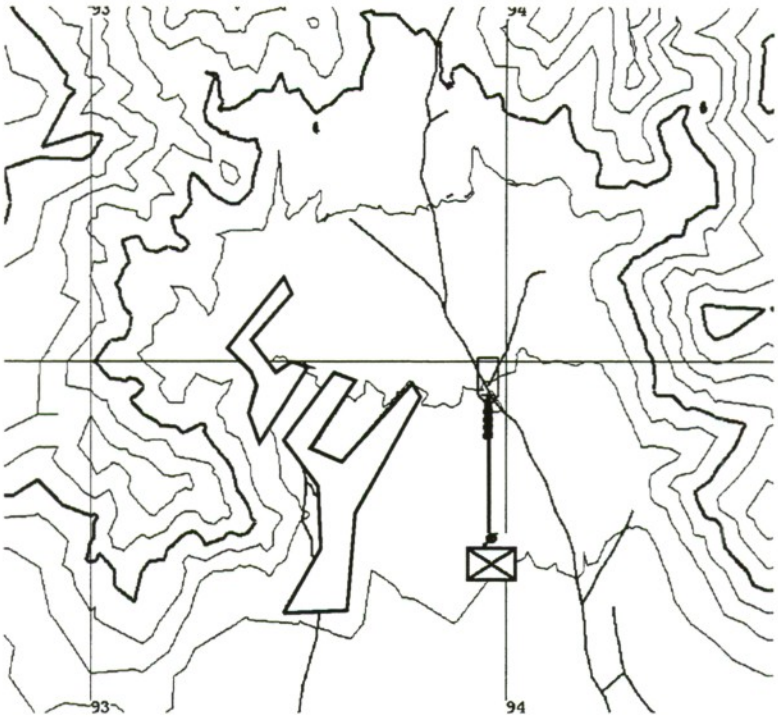


Figure 3: Results of planning "shortest-path" from unit's position to its goal, for a case in which the resulting path does *not* use the concealed regions: The algorithm has discovered that deviation of the straight-line path to use the available concealment would add more exposure (for the sections of the route that carry the unit to and from the concealed regions) then it would remove by utilizing those regions.

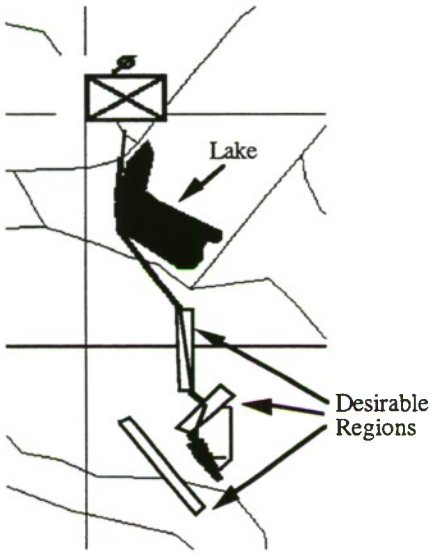


Figure 4: A path which both utilizes desirable regions and avoids obstacles. The unit must detour around a lake to reach its destination. It uses two of three low-cost regions in its minimal path. While this example utilizes only three different numerical region weights (as shown in Figure 1), the weighted regions approach allows the assignment of a whole spectrum of weights, to specify graded levels of desirability for the terrain regions.

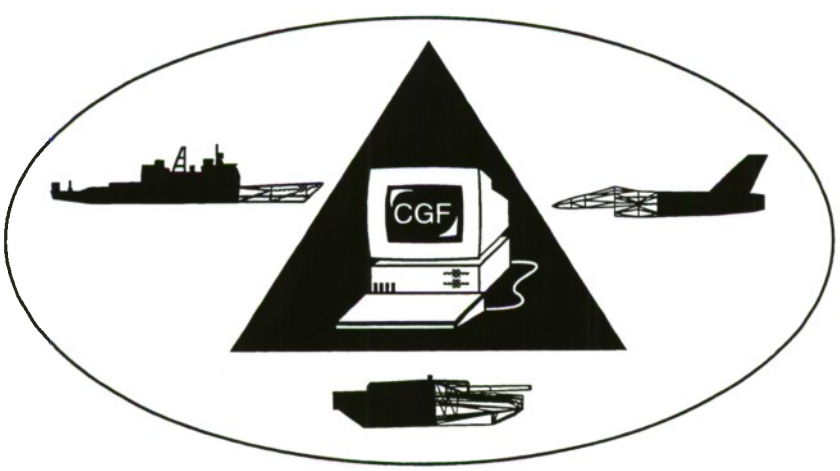
Distributed Simulation, Cambridge, Massachusetts.
Mitchell, J.S.B. (1988) "An Algorithmic Approach to Some Problems in Terrain Navigation", Artificial Intelligence, 37: 171-201.
Mitchell, J.S.B., Papadimitriou C.H. (1991) "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision", Journal of the Association for Computing Machinery, 38(1): 18-71.
Rowe, N.C., Ross, R.S. (1990) "Optimal Grid-Free Path Planning Across Arbitrarily Contoured Terrain with Ansiotropic Friction and Gravity Effects", IEEE Transactions on Robotics and Automation, 6(5): 540-553.

10. Authors' Biographies

Bruce Hoff is the Principal Investigator for the USMC-SAF project at Hughes Research Laboratories. At H.R.L. Dr. Hoff has also worked on the ARPA autonomous vehicle project, and CGF Command Forces (C-FOR). Dr. Hoff earned B.S. and M.S. degrees from Rensselaer Polytechnic Institute, and the Ph.D. degree from the University of Southern California, all in Computer Science. His research interests are in the areas of Artificial Intelligence, Autonomous Vehicles, Adaptive Control, and Computer Generated Forces.

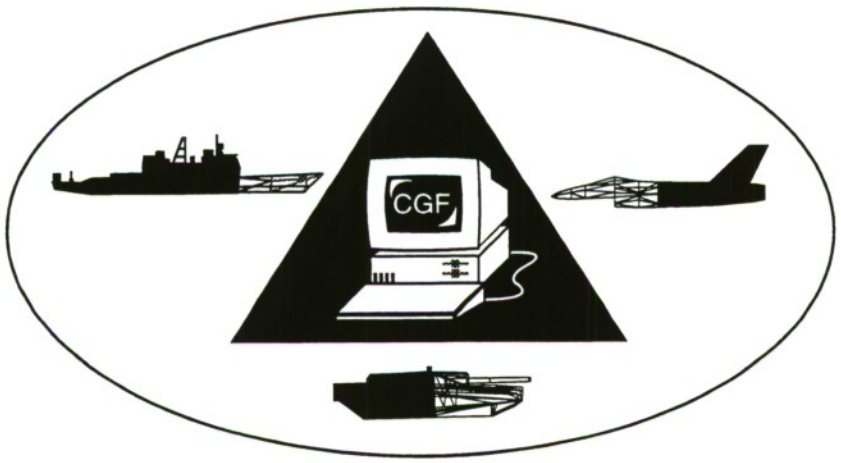
Michael D. Howard has been active in the study of Semi-Autonomous Forces and Autonomous Systems since 1988. Since the work described in this paper, he has become Principal Investigator of the Command Forces program, a new ARPA program. He continues to contribute to the USMC SAF program. Mr. Howard holds the MSEE degree from University of Southern California (1986), a BSEE from Louisiana State University (1981), and a BA in English from St. Andrews College (1977).

David Y. Tseng joined Hughes Research Laboratories in 1969. For the past 15 years, he has been actively involved in directing research in autonomous systems, distributed simulation, and information management systems. In 1988, he started Hughes' research activities in autonomous agents, which paved the way for the current SAF and CGF research, and remains one of his primary interests today. He is responsible for initiating and directing the Hughes autonomous vehicle navigation program which culminated in the first autonomous cross country navigation in 1987. In 1984, he was instrumental in initiating the Artificial Intelligence activities in Hughes, and was manager of the Hughes AI Center for 10 years. He received his Ph.D. from the Polytechnic Institute of Brooklyn, S.M. from Harvard Univ., and B.S.E.E. from the Univ. of Pennsylvania. He is a member of IEEE and AAAI, has 29 technical publications, and holds 3 patents.


CGF

Session 6a: Implementation

Courtemanche, Loral ADS
Ourston, SAIC
Pratt, NPGS


CGF

Representation of Missiles in ModSAF

Anthony J. Courtemanche, Scott E. Hamilton, and Paul Monday
Loral Advanced Distributed Simulation
50 Moulton St., Cambridge, MA 02138
ajc@camb-lads.loral.com
shamilto@camb-lads.loral.com
pmonday@vox.knox.loral.com

1. Abstract

The DIS standards allow for missiles to be represented as visible entities in the synthetic battlefield. One approach to modeling missiles in DIS is to treat them as full dynamic entities, with explicit simulation of the propulsion, guidance, and fuzing of the missile prior to its detonation. A model such as this will typically require a great deal of fidelity and processing power to accurately characterize the missile. A second approach is to use a statistical method, as is frequently done for direct fire weapons.

This paper describes two possible implementation approaches to modeling missiles in DIS, and how ModSAF has provided a working testbed to implement both types of models. The fundamental advantages and disadvantages of each approach are discussed, and the implementation details in ModSAF are described.

2. Use of Missiles In ModSAF

The DIS standards allow for missiles to be represented as visible entities in the synthetic battlefield. These missiles are similar to other simulation entities in that they generate Entity State PDUs and can move and interact with other entities in the environment. There are subtle differences, however, between missile entities and other types of entities. One difference is that missiles are usually launched from DIS vehicles. As such, the missile is dormant until the time of launch, at which point the missile appears near its launcher and subsequently flies toward its destination. Once it has arrived at its destination, the missile detonates and is removed from the DIS exercise. A second difference between missiles and other entities is that missiles typically move at much greater velocities than other entities in the DIS simulation. A third difference is that relatively minor errors in the modeling of vehicle dynamics can have a large effect on the battlefield, since they can cause large errors in the effective probability of hit of the missiles.

There are several reasons why it is useful to represent missiles as visible entities in a DIS environment. First, the observation of visible missile flyout can provide clues about the location of the enemy vehicle that is shooting the missile. Second, targets can attempt to defeat incoming missiles via evasion and countermeasures. These can only be attempted if the target can detect the incoming missile in some way. Third, visually displaying dynamic missile entities allows weapon system developers and simulation modelers to visualize and understand missile guidance and dynamics models.

The ModSAF, Computer Generated Forces (CGF) system (Ceranowicz 1994), supports a number of different types of missiles. There are anti-tank (AT) missiles which are both ground and air launched. There are also anti-aircraft (AA) missiles which are both ground and air launched. Each of these missiles could be implemented in ModSAF using a dynamic or probabilistic simulation.

Missile	Type	Delivery	Dynamic / Probabilistic
Alamo	AA	Air	D
Archer	AA	Air	D
Dragon	AT	Ground	D
Gaskin	AA	Ground	D
Gauntlet	AA	Ground	D
German HOT	AT	Ground	P
USSR HOT	AT	Air	D
Hellfire	AT	Air	P
Javelin	AT	Ground	P
LOSAT	AT	Ground	P
Maverick	AT	Air	D
Milan	AT	Ground	P/D
NLOS TV	AT	Ground	P*
Phoenix	AA	Air	D
SA-16	AA	Ground	D
SA-19	AA	Ground	D
Sagger	AT	Ground	P
Sidewinder	AA	Air	D
Songster	AT	Air/Ground	P/D
Spandrel	AT	Ground	P
Sparrow	AA	Air	D
Spigot	AT	Ground	D
Spiral	AT	Air	D
Stinger	AA	Air/Ground	D
TOW	AT	Ground	P/D

Table 1: ModSAF Missile Types

Table 1 contains the missiles which are currently implemented in ModSAF, what type of missile each

is (Anti-Tank or Anti-Aircraft), and from where the missiles are delivered. The fourth column shows how the missiles are currently being implemented, where D indicates a dynamic missile simulation, P indicates a probabilistic simulation, and P/D indicates that both dynamic and probabilistic simulations are implemented.

A vehicle parameter file currently exists in ModSAF for each of these missiles, so all of them could be implemented as dynamic missile simulations as explained in section 3. Also, as will be described in section 4.2, there are default flyout equations that could be used to support a probabilistic simulation, so all of these missiles could also easily be implemented as probabilistic missile simulations.

The one exception is the NLOS TV missile which is implemented via another library which relies on the ballistic gun model to provide the probabilistic missile simulation (hence the P* in table 1). A vehicle parameter file does not currently exist for this missile.

The Dynamic and P_h (probabilistic) Missile Simulations will be explained in more detail in the following sections.

3. Dynamic Missile Simulation

Since ModSAF Version 1.0, released December 1993, ModSAF has had a dynamic missile simulation. This simulation was originally developed for the weapons of threat and friendly aircraft in support of the WISSARD project (Rosenbloom et. al. 1994). The simulation contains a motion model of the boost and coast phases of the missile, as well as a collision model and detonation model. Some of these missiles, such as the Phoenix missile launched from the F14D aircraft, are sequenced by complex state machines which control the interaction between the sensors on the firing aircraft and the missile itself. For example, the Phoenix missile can be controlled by the host aircraft for the majority of its flight. As the aircraft continues to track the target with its on-board radar system, the aircraft will issue steering commands to the missile to have it fly toward the target. During the last phases of the engagement, the missile can be commanded to go into an active mode, where an on-board radar system on the missile can track the target itself, without any commands from the host aircraft.

In order to use Dynamic Missile Simulation in ModSAF, the launching vehicle must contain a missile launcher model which shoots the missile. A vehicle parameter file must exist for the missile munition which describes its flight dynamics.

3.1 Equations of Motion

The dynamic missiles in ModSAF are controlled by the following equations of motion, where $\vec{P}$ is the position of the missile, $\vec{v}$ is the velocity of the missile, τ is the time between the last tick and the current tick, $\vec{D}$ is the desired missile direction clamped by a maximum rate of turn, S is the current speed of the missile, clamped by a maximum speed, and A is the acceleration of the missile during its boost phase:

$$\vec{P}_{New} = \vec{P}_{Old} + \vec{v}_{Old} \cdot \tau$$

$$\vec{v}_{New} = S \cdot \vec{D}$$

$$S = A \cdot \tau$$

There is no effect of gravity, unless the missile has lost power, in which case it falls along a parabolic trajectory.

3.2 Collision And Detonation Modeling

As a dynamic missile moves through space, there are two types of events which may affect it. First, the missile can collide with another object or the terrain. The ModSAF library LibCollision provides a 3D physical model of collision detection. It can detect collisions with other network entities such as vehicles, other missiles, or even dynamic structures, as well as terrain objects such as treelines, buildings, and the ground. This library is also responsible for generating and processing Collision PDUs. In the case of colliding with other entities, collision detection for the missile is based on the intersection of the bounding boxes of the missile and the object being collided with.

A second event that may affect a missile during its flight is detonation near a target. The ModSAF library LibDetonation provides a model of proximity detonation. It can detect detonations due to proximity with other network entities This library determines that a detonation should occur if the distance to the target, as measured from the secant between positions of the missile during consecutive ticks, achieves a local minimum and is less than the detonation radius parameter specified for the missile. In performing this calculation, the position of the target is projected forward or backward in time to find the point on its trajectory closest to the point where the local minimum occurred.

If either a collision with an entity or a proximate detonation with an entity is detected, a DIS Detonation PDU will be sent specifying the target entity. In the case of a detonation, the DIS

Detonation result type will be "PROXIMATE DETONATION". In all other cases, the result type will be "IMPACT".

3.3. Missile Guidance

The modeling of the seeker and guidance of the missile can vary from missile to missile. For example, the Phoenix missile can be steered by the radar sensor on the host aircraft, or by its on board radar sensor. The TOW missile is wire guided and could also be modeled as being guided by a visual sensor on the host platform, in this case the Bradley Fighting Vehicle. A fire-and-forget missile such as the Stinger missile can be guided exclusively by its on-board IR sensor.

Lead-pursuit and pure-pursuit guidance is supported, and all guidance information simply generates the desired direction, $\bar{D}$, for the missile.

3.4 Advantages

The advantage of this implementation is that it is an interactive simulation. As the target changes location, the missile can adapt dynamically. This allows for the possibility of simple counter-measures, such as target jinking, to attempt to defeat the missile.

3.5 Limitations

There are two major types of limitations to the dynamic missile approach in ModSAF. First is the rather simplistic modeling of the missile, as represented by the equations of motion. The various forces, such as thrust, gravity, and drag are not modeled. The treatment of turn rate is very simplistic. Maximum turn rate should depend on speed and altitude. Related to the simplicity of the model is the inability to predictably control missile performance. The only real performance inputs to the dynamic missile model are maximum speeds, burn times, and turn rates. This is an insufficient amount of control to distinguish different types of missile performance. Especially lacking is the type of guidance control that would allow a dynamic ModSAF missile to approximate the trajectory of an actual missile (Bencke et. al. 1994). Also, it is impossible to correlate this simple missile model with available field testing data, such as probability of hit. This makes it difficult to use these types of missiles in a combat developments experiment.

The second type of limitation to this missile approach in ModSAF is the tick rate constraint that ModSAF must obey. Available computational resources are always a limiting factor in the number of entities that a ModSAF workstation can support. ModSAF is a variable frame rate simulation. Ground vehicles in ModSAF have been tested to behave acceptably under a 2 Hz frame rate. This frame rate characterizes the minimum acceptable simulation performance (Vrablik & Richardson 1994). Obviously, a varying frame rate that can be as low as 2 Hz is quite limiting for the modeling of a high speed missile. ModSAF does support high priority scheduling which could be used to guarantee a faster frame rate for missiles, however it would be difficult to accept the resulting loss of performance in the rest of the system.

4. Ph Missile Simulation

The dynamic missile model has proven to be unsatisfactory for combat developments experiments such as the Anti-Armor Advanced Technology Demonstration (A2ATD). When DIS is used for combat developments, there are many issues that must be resolved when running experiments involving missiles and other weapons fire. For example, the algorithms used to model the accuracy of a given weapon system against a particular target must be verified and validated before useful experimental results can be gathered. As the dynamic missile model fails to generate accurate flight trajectories and cannot be controlled to give specific performance results, it cannot be used for this type of experiment. A different approach, which is based on a statistical model of accuracy, is required.

4.1 Direct Fire Biases & Dispersions Delivery Accuracy Model

For non-guided direct fire weapons, Army Materiel Systems Analysis Activity (AMSAA) has prescribed a statistical model based on biases and dispersions to characterize the accuracy of direct fire weapons. This model generates a horizontal and vertical miss distance for each round at the target's location, referenced to an assumed aim point (Courtemanche & Monday, 1994).

The ModSAF library LibBalGun provides a model of a ballistic gun's operations with the delivery accuracy portion based on biases and dispersions. It includes modeling of the time from gunner initiation of the engagement to firing (track time), gun firing and loading, application of biases and dispersions to generate an impact point, and determination of hit or miss.

A vehicle level task identifies the most urgent target and makes recommendations of which weapons to use against the target. A vehicle level task collects the recommendations and performs targeting actions against the target. Part of these targeting actions is to give the target to a gun so it will fire at it. It is

possible for the gun to determine that it will actually hit another target or location based on the lack of intervisibility to the original target. The gun may also be given a location to fire at instead of a target. This functionality can be used to launch smoke grenades or to shoot at a laser designation.

Range indexed track time tables are used to define how long the target will be tracked before the gun is fired. The track time is generated from these tables using a value from a lognormal distribution which is adjusted for firer competency and firer and/or target motion.

The gun determines whether the target was hit or missed using the biases and dispersions generated from hit tables. These tables are indexed by range and contain biases, error, and firer and target stationary/motion dispersion values. The hit tables are derived directly from AMSAA weapon delivery accuracy data (Topper, 1993). The biases and dispersions define the impact's offset from the aim point. The aim point's actual location is calculated from the target's location when the munition arrives at the target. This approach effectively removes target behavior from the outcome and ensures that the desired delivery accuracy statistics are honored. If the target was hit the gun sends a Detonation PDU on the target, otherwise it sends a Detonation PDU on a miss location at impact time.

P_h missile simulations are initiated or shot from these same direct fire ballistic gun models in ModSAF. These simulations do not use missile dynamics for determining whether or not the missile will arrive at the target. Instead the probability of hit is determined from the biases and dispersions. In order to use the P_h missile simulation, the launching vehicle must contain a ballistic gun model which shoots the missile as a munition.

This method of missile delivery was first used in A2ATD experiment 1 replication of the M1A2 IOTE in order to achieve the desired hit probabilities for the Sagger missile launched from the T-80 tank and the Spandrel missile launched from the BMP2. Before this, the missiles were delivered as Dynamic Missile Simulation shot from missile launcher models. Delivering the two missiles via a P_h Missile Simulation entailed adding hit tables for the Sagger and Spandrel delivery accuracy model and using ballistic gun models to shoot the munitions. There were no visual flyouts of the missiles via the P_h Missile Simulation since, at the time, ballistic guns only sent out Fire and Detonation PDUs.

4.2 Extensions to support missiles

In order to address the lack of a visual missile flyout when the munition was shot from a ballistic gun as a P_h Missile Simulation, extensions were made to ModSAF.

The ballistic gun model has been enhanced to shoot munitions with or without visual flyout. Visual flyouts are accomplished via the addition of a computationally lightweight missile entity. The ballistic gun model is responsible for ticking the lightweight missile entities. The tick processing causes Entity State PDUs to be sent out for the missile entities. These PDUs provide for the visual flyout. This capability can be used for any munition, although it is most appropriate for slow-moving missiles.

The actual positions of the missiles during the flyout are provided by flight path or flyout equation functions. The ballistic gun provides a service which allows flyout equation functions to be registered with the gun. As part of the tick of the lightweight missile entities, the gun calls the registered flyout equation function to update the missile position and provide other information such as whether or not the missile has reached the target. Each munition can have a separate flyout equation registered, or the same flyout equation can be used for multiple munitions.

As described previously, the ballistic gun is responsible for determining whether or not the missile actually hits the target based on the statistical outcome of the biases and dispersions. If the target is not hit, the flyout equation function is also responsible for calculating and flying out to a miss location. The ballistic gun model also makes sure that the missile position provided by the flyout equation avoids the terrain until the missile has reached the point at which it should impact, as shown in Figure 1.

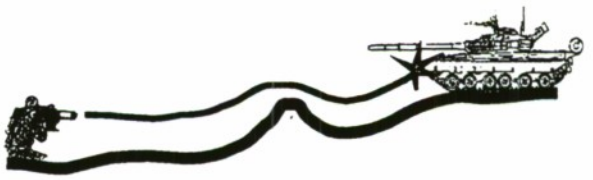


Figure 1: Javelin Missile Engagement

Besides providing updated positions, the flyout equation can also change from flying at a target to flying to a location or vice versa. The first capability can be used to model breaking missile lock on a target due lack of intervisibility, as is performed in the CASTFOREM model (Mackey et. al., 1994). The second capability can be used to model a non-

line-of-sight missile flying to a location prior to acquiring a target.

4.3 Advantages

There are a number of advantages to the P_h Missile Simulation in comparison to the Dynamic Missile Simulation. The probability of target hit is controlled which is crucial for the validation of models. These probabilities are generated from easily-available and understood data. Another major advantage is that realistic flight-paths can now be easily incorporated via the flyout equation function registration. The inherent flexibility of the flyout equations and registration service for modeling new systems has already been demonstrated, as will be described in section 5.3.

4.4 Limitations

There are some limitations to the P_h Missile Simulation. These limitations are due to the statistical nature of the simulation as well as the fact that the P_h model will typically guarantee delivery of the missile to target.

The P_h Missile Simulation is inherently non-interactive. This is an unavoidable consequence of honoring a statistical delivery accuracy model. If evasive battlefield behaviors like jinking are expected, the delivery accuracy statistics can be adjusted accordingly. But since this adjustment will be for the typical case, it will not match each specific case. So the target that makes a better-then-expected jink will not necessarily be rewarded.

Because of the requirement to honor delivery accuracy statistics and because the evaluation of these statistics is performed when the missile reaches the target, some visual anomalies are sometimes apparent. For example, missiles never impact the ground significantly short of the target. In fact, missiles are prevented from hitting the ground even if their flyout would otherwise cause that event. Otherwise, the delivery accuracy statistics would be unduly distorted. Similarly, even if missiles appear to fly through such obstacles as buildings, trees, or other vehicles, they do not detonate but instead continue on to their original target.

There is no current ability to deal with counter measures. The lightweight missile entities do not contain all the models that would be required and neither the ballistic gun models nor the flyout equations are equipped to handle these situations. However, the effects of counter-measures could be added in the future when required.

5. Remotely Designated Missiles

The use of the P_h methodology to represent missiles has proven to be very useful for specialized missiles such as SAL (Semi-Active Laser) Hellfire. The SAL Hellfire missile can be launched from an attack helicopter such as an AH-64 Apache. The SAL Hellfire missile seeks the reflection of a laser designation spot on a target. Either the firing Apache can designate the target, or the target can be designated by a remote vehicle such as a OH-58D Kiowa scout helicopter. The remote designation case is the most complex, as described below.

5.1 Remote Laser Designation Process

Figure 2 shows a typical scenario for remote designation with the Hellfire. Here, a scout aircraft has line of sight to the target and designates it by shining the laser on it. The attack aircraft can fire from a concealed position, and the Hellfire will track toward the laser spot once the missile acquires the spot with an on board sensor. The advantages to this technique are that the scout can lase from one azimuth, with potentially only its mast laser device exposed, while the attack helicopter is completely concealed along a different azimuth. Since the missile firing is from a concealed position, very little visual cueing is provided to the target. Both the scout and the attack helicopter may have increased survivability when remote laser designation is used.

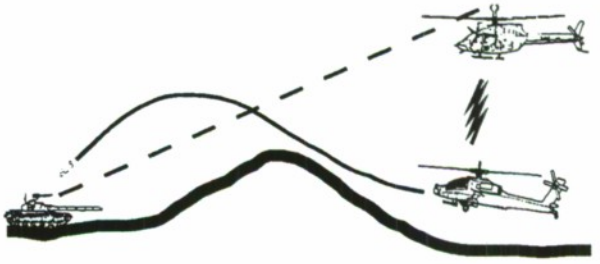


Figure 2: Remote Laser Designation

5.2 LDWSS Model

The Laser Designator Weapons System Simulation (LDWSS) system is used by AMSAA to relate laser designator performance to weapon system performance (Alongi et. al. 1984). The output of LDWSS simulations are performance tables which relate round center of impact and dispersions to target range for a given combination of designation (scout lasing or attack helicopter firing and lasing), tracking sensor (TV, FLIR, DVO), and tracking mode (manual, automatic). For A2ATD experiments involving Hellfire missiles, it is desirable that ModSAF be able to use LDWSS data as part of its

Hellfire simulation. Under close supervision by AMSAA, the following methodology has been developed to support remote laser designation by both ModSAF and the manned Longbow and Commanche simulators that are participating in the exercise. This methodology is a natural extension to the existing P_h missile methodology.

5.3 ModSAF Implementation

The methodology chosen for implementation of SAL Hellfire in ModSAF decouples the components of the designator and firer as much as possible. Under this methodology, the designator determines laser spot position on the target by drawing from a random number generator based on test data. This data is indexed by tracking sensor and tracking mode, but this is all internal to the designator, whether the designator is a manned simulator or a ModSAF vehicle. This data is not summary output LDWSS performance data, but will be available from AMSAA as test data or data that can be computed from the SPOTGEN (Spot Jitter Generator Model) subcomponent of the LDWSS simulation. The output of this random draw will be a spot location, relative to the target. The randomization that is done is very similar to the biases and dispersions calculation done for direct fire, however the input tables may be in a slightly different format.

The spot location is broadcast via DIS Laser PDUs. As the spot location is broadcast in DIS, all exercise participants can react to the presence of the spot, if they can detect it with the appropriate sensor. Actual laser designation spots have encoded information in them to allow unique designation of multiple targets in the same area. In DIS, the designator will specify a laser code for the laser spot using the laser code field in the Laser PDU, and the SAL Hellfire missile will have to be seeking that same laser code for it to engage and track that particular laser spot.

The spot location incorporates the error of the spot with respect to the aim point on the target. The firer of the Hellfire missile, who may be the same entity as the designator, or may be a different entity, will use "perfect aim" LDWSS data to deliver the round to the spot location. This perfect aim LDWSS data has no component of spot location error in it. This perfect aim data simply describes the accuracy of the missile in hitting the spot. This perfect aim data is completely independent of the sensor device or tracking mode used by the designator to generate the spot. The firer draws from a random number generator based on the perfect aim LDWSS data to determine the error of the missile in hitting the spot. This randomization is also very similar to the biases and dispersions calculation done for direct fire, with a slightly different input table format as generated by the LDWSS model. At the time of the arrival of the missile to spot location, the firer will add the two errors together to determine the actual location of the missile with respect to the target, and therefore determine a hit or a miss, as shown in Figure 3.

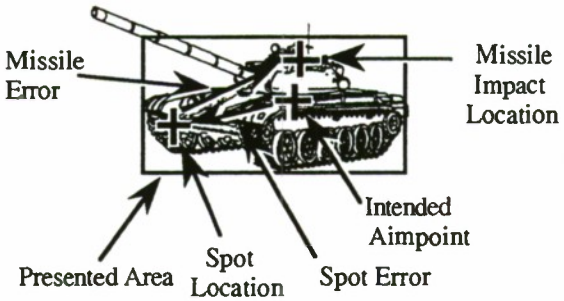


Figure 3: SAL Hellfire Delivery Accuracy

In this diagram, we see that the error of the missile impact with respect to the spot, which was drawn by the firer, is added to the error of the spot with respect to the aim point on the target, which was drawn by the designator and broadcast as part of the information in the Laser PDU. The sum of the two errors is used in geometric calculations to determine whether the missile actually intersected the target. If an intersection is determined, a Detonation PDU which contains a relative impact location with respect to the target will be issued.

This methodology allows for the laser designation spot to move or disappear from the battlefield while a missile is in flight. This can occur if some situation forces the designator to alter what it is designating. During flight, the missile tests for line of sight with any of the laser spots that are encoded with the proper laser code. If line of sight is established, the missile will adjust its flight path to seek toward that laser spot. During the time of flight, the Hellfire missile issues Entity State PDUs according to a registered flyout equation as specified in section 4.2. This equation allows the missile to trace a path that is representative of an actual Hellfire missile.

This methodology allows the designator and the firer to be different entities, or the same entity. If the firer and designator are different entities, this methodology also allows any combination of manned and ModSAF vehicles to take on the roles of firing vehicle and designating vehicle.

Since, in the case of remote designation, the firing vehicle is not required to have line of sight to the target before engaging, some sort of command and control is required to coordinate the scout lasing with

the attack helicopter firing. In an initial implementation, this is accomplished in ModSAF via the transmission of an ASCII encoded radio message from the scout to the attack helicopter. This message contains information about the location of the target and that the target is ready to be engaged. Future implementations may make use of specialized digital messages defined by the Air Warfighting Cell (AWC) for transmission of targeting information between scout RAH-66 Commanche helicopters manned simulators and attack AH-64D Apache Longbow manned simulators.

6. Future Work

6.1 V&V by AMSAA

The P_h missile model will be evaluated for Validation and Verification (V&V) by AMSAA as part of the A2ATD program. AMSAA has already evaluated the delivery accuracy model LibBalGun for direct fire weapons. This work was performed by causing ModSAF vehicles to fire at targets many times and recording a variety of internal model parameters for each firing using the Direct Fire Delivery Accuracy VVA Data structure encapsulated in a DIS Event Report PDU. This data was analyzed to verify such processes as table lookups, handling of biases and dispersions, determination of aim point, calculation of miss distance, and hit assessment.

Since the P_h missile model is mainly an extension of LibBalGun, the V&V evaluations can be performed in a similar manner.

6.2 Alternate P_h Mechanisms

ModSAF currently supports the standard direct fire bias and dispersion tables or the LDWSS accuracy tables to compute hit or miss based on a geometric model of the target and a randomly calculated error from the desired aim point. As such, the calculation of hit or miss is derived primarily from a geometric calculation, and an explicit probability of hit (P_h) is not actually represented in the model. Other statistical methods might also be desired. For example, AMSAA and TRAC studies involving the Javelin missile have traditionally used test data that provide a simple P_h based on target range, target type, and target aspect angle. Although not currently in place, it would be straightforward to extend the existing ModSAF methodology to use one of a number of different statistical or geometrical calculations to determine hit or miss, while still supporting flyout equations to generate the missile flight paths. Based on current direction from AMSAA, all ModSAF missile flyouts for A2ATD experiments will be based on the statistical biases and dispersions methodology.

6.3 Engagements of NLOS Missile

Analysis of upcoming A2ATD scenarios shows that additional behaviors involving missiles may be required. For example, vulnerability data will be provided for the NLOS missile to be potentially shot down by the 2S6 air defense vehicle. This will require a number of changes. For instance, the 2S6 firing behaviors will have to be enhanced to target and engage missiles as well as aircraft. More importantly, the NLOS missile simulation will have to be able to accept received DIS Detonation PDUs and calculate damage based on the information in those PDUs. This should be a straightforward enhancement. Just as every regular vehicle in ModSAF has damage processing mixed in to the vehicle via the including of the LibDFDam (Direct Fire Damage) vehicle subclass as part of the vehicle definition, the lightweight entity used to represent the NLOS missile will need to include this subclass as well.

6.4 Missing Missile Behaviors

As a side-effect of moving the launching of missiles from missile launchers into ballistic guns when using the P_h Missile Simulation, certain launcher model capabilities do not occur during the simulation. For example, using the Dynamic Missile Simulation to shoot a TOW missile from an M2 in ModSAF causes the M2 to include an appearance bit in its Entity State PDU which will show the TOW launcher as being in a raised position. This capability was not implemented in the ballistic gun model because guns did not have corresponding launcher up/down positions. This deficiency will be corrected in future ModSAF versions.

6.5 Generalization

While implementing the P_h extensions to ModSAF, efforts were made to reuse common models and provide for future reuse. The biases and dispersions model was abstracted out of LibBalGun and put in a combat models library for reuse by laser designation or other models. The ballistic gun model is heavily used in ModSAF and historically has been modified repeatedly as new munitions and systems have been added. This model needs to remain fairly stable for verification and validation purposes. By providing the flyout equation registration service, new flyout equation functions can be added as opposed to modifying LibBalGun as new munitions and systems are added. It is likely that future development in ModSAF will turn up other opportunities to

generalize existing simulation software for the purpose of reuse.

7. Conclusions

Two implementation approaches to modeling missiles currently exist in ModSAF, Dynamic Missile Simulation and P_h Missile Simulation. Each approach has its advantages and disadvantages, however the P_h Missile Simulation approach is needed for combat developments experiments because of the ability to control the outcome. In addition the P_h method is easily extended and adapted for new munitions and systems, further adding to its usefulness for experimentation and system validation.

8. Acknowledgments

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058.

9. References

Alongi, R. E., Bosley, J., and Lee, A. W. Jr. (1984). "LDWSS Users Guide", *Technical Report RG-84-5*, US Army Missile Command, Redstone Arsenal, Alabama.

Bencke, J., Mosier, .P., Marks, B., Chavin, S. (1994). *Technical Report - Studies/Services, Software Component Test Report (DRAFT)*, Illgen Simulation Technologies Inc.

Ceranowicz, A. (1994). "ModSAF Capabilities", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 3-8/

Courtemanche, A. J., and Monday, P. (1994). "The Incorporation of Validated Combat Models into ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 129-140.

Mackey, Dixon, Jensen, Loncarich, Swaim (1994). *CASTFOREM Update: Methodologies*, Department of the Army, US Army TRADOC Analysis Command.

Rosenbloom, P. S., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Lehman, J. F., Rubinoff, R., Schwamb, K. B., and Tambe, M. (1994). "Intelligent Automated Agents for Tactical Air Simulation: A Progress Report", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 69-78.

Topper (1993). *A Compendium of Close Combat Tactical Trainer Data Structures, Algorithms, and Generic System Mappings*, AMSAA.

Vrablik, R., and Richardson, W. (1994). "Benchmarking and Optimization of ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 25-33.

10. Biographies

Since graduating from MIT with a S.M. in Electrical Engineering and Computer Science in 1987 and a S.B. in Electrical Engineering in 1986, Mr. Courtemanche has worked in the Multiprocessor Lisp group at Bolt Beranek and Newman (BBN) and in the SAF group at Loral Advanced Distributed Simulation (formerly BBN Advanced Simulation). At Loral, he is a Senior Software Engineering Specialist and has been Project Engineer for the ModSAF System Development Delivery Order. Mr. Courtemanche has been a major contributor to the architecture, protocols, weapon systems modeling, and targeting behaviors in ModSAF, Odin, TCE, and SIMNET SAF. He works out of the Loral ADST office in Orlando, Florida.

After receiving his B.S. in Computer Science from Brigham Young University in 1985, Mr. Hamilton worked for Hughes Aircraft Company until joining Loral Advanced Distributed Simulation in January of 1994. Mr. Hamilton was a Hughes Masters Fellow and received his M.S. in Computer Science from the University of Colorado at Boulder in 1993. At Hughes, Mr. Hamilton designed and developed large-scale satellite ground station systems. While there, he was a member of a small team which began Hughes' successful object-oriented C++ development. At Loral in Cambridge, Massachusetts, Mr. Hamilton is a Senior Software Engineer designing and developing software for the ModSAF program.

Mr. Monday has worked on the SIMNET-D and ADST projects for BBN and Loral since 1987. He is currently Chief Analyst at the Mounted Warfare Test Bed (MWTB), Ft Knox, Kentucky where he designs and develops software for data analysis, ModSAF, and other simulations. He graduated from the University of Toledo in 1978 with a B.S. in geology and from Stanford University in 1979 with a M.S. in geophysics. Mr. Monday previously worked for 7 years in petroleum exploration.

From CIS to Software

Dirk Ourston, David Blanchard, Edward Chandler, Elsie Loh
Science Applications International Corporation
3045 Technology Parkway, Orlando, FL 32826

Henry Marshall
U.S. Army STRICOM

1. Abstract

This paper presents the approach used on CCTT SAF to transform tactical behaviors into delivered software. The process starts with the development of a Combat Instruction Set (CIS), generated by Subject Matter Experts (SMEs). This natural language description of the tactical behavior is then transformed into detailed software requirements through a process that was developed on the CCTT SAF program. The detailed software requirements are then implemented as Ada software using the DOD–STD–2167A methodology.

2. Introduction

One of the challenges facing the builders of the Close Combat Tactical Trainer (CCTT) Semi–Automated Forces (SAF) simulation is that of generating SAF behaviors that can be efficiently implemented in software, and yet accurately reflect tactical doctrine. To accomplish this, a new type of behavior description has been created, a Combat Instruction Set (CIS) [4]. Through a series of well–defined processes the CIS is translated into simulation software. The completed CISs serve as both a tactical data base for any future tactical simulations requiring combat behaviors, and the source of the behavior specifications for the CCTT SAF. This paper describes the process by which a CIS is developed and transformed into implemented software.

3. The CIS–To–Software Process

Figure 1 shows the process used for implementing software derived from CISs. The figure divides the software development process into five phases, CIS development, CIS software analysis, preliminary design, detailed design, and code and development test. Figure 1also shows the "interface" documents for the software development process, that is, those documents that serve to connect one phase with the next. The phases "CIS Development" and "CIS Software Analysis" correspond to the software requirements analysis phase in traditional software development. Because these phases are unique to the CIS software development activity, they are the focus of this paper.

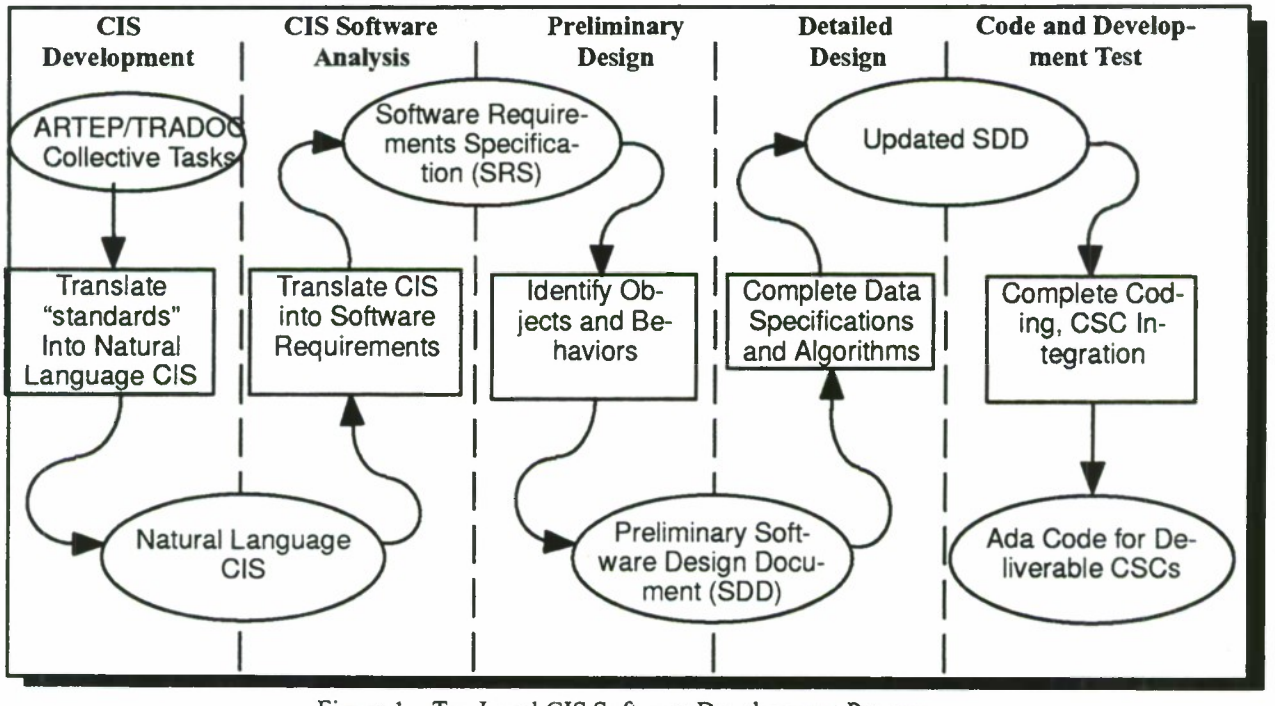


Figure 1: Top Level CIS Software Development Process

Figure 2 presents more detail concerning the products generated during the software development process. As can be seen from the figures, the CIS development activity starts with reference doctrinal literature and results in a Natural Language CIS. The natural language CIS provides a structured description of and due to space limitations are not covered in more detail here. The interested reader is referred to reference 2.a combat behavior The details of the CIS development activity are presented in Section 4.

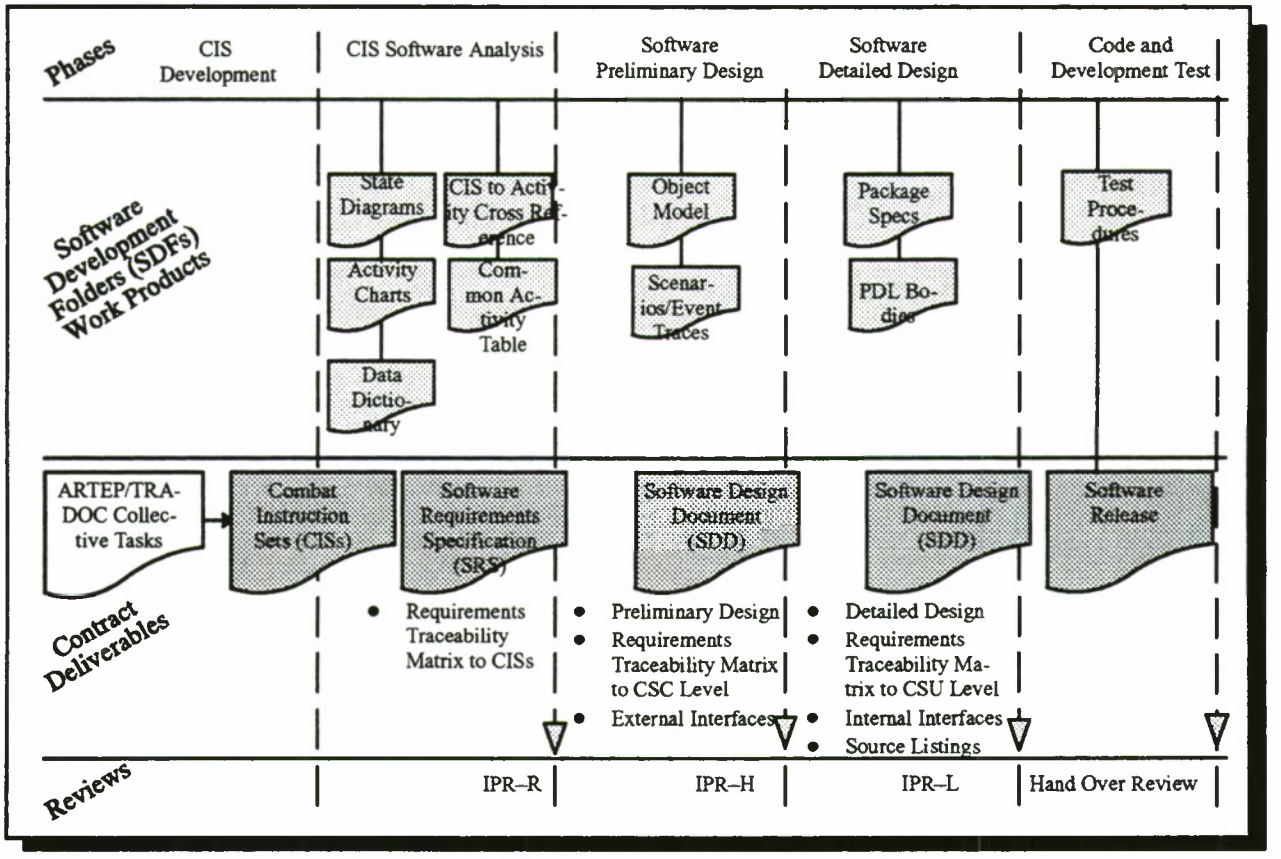


Figure 2: CIS Analysis and Implementation

The natural language description is then used as the starting point for the CIS software analysis activity, described in Section 5. The CIS software analysis activity results in a Software Requirements specification that is then used as the basis for the remainder of the software design and development activity. For CCTT SAF, these software design and development activities follow the DOD–2167A standard as interpreted for the CCTT program ([1], [2]), and reported on in [5]. The standard requires that the developer maintain Software Development Folders (SDFs), which contain development and test information relating to the various software components. The information in the folders are used to build the various documents required. Many of the CIS process products generated reside in the SDFs. An automated requirements tracking tool, Requirements Traceability Matrix (RTM) Tool is being used to track traceability from ARTEP to code. Due to space limitations, for a detailed implementation of the DOD–2167A standard, refer to reference 2.

4. CIS Development Description

This section provides further detail on the process used to develop the CIS, and the products that result.

4.1 CIS Development Process

The natural language Combat Instruction Set (CIS) is obtained through the conversion of current training and doctrinal literature into a structured, formatted description of how a training task is to be performed. The natural language CIS therefore forms the description of interactions between platforms and units that software (S/W) engineers will implement in code. A CIS describes tactical combat behavior at the unit and platform level. After translation into software, CISs may be utilized by SAF Operators to emulate specific unit and platform behaviors in support of the execution of CCTT operational plans.

For BLUFOR units, the CISs are based on current doctrine and tactics found in U. S. Army Training and Evaluation Program (ARTEP) Mission Training Plans

(MTPs). For OPFOR units, the CISs are based on foreign military publications and the U. S. Army Training and Doctrine Command (TRADOC) OPFOR Heavy Guide. In addition, doctrinal references are consulted for OPFOR behaviors, since training standards do not exist for OPFOR units. In developing CISs, behaviors are organized according to four major primitives (move, shoot, search/observe, and communicate) and include initial conditions, termination conditions, and situational conditions (interrupts) that may trigger the execution of other CISs. All developed CISs require formal approval. This is provided by responsible Army schools for BLUFOR doctrine, and TRADOC's Threat Support Division for OPFOR doctrine.

The process for developing doctrinally correct CISs was created jointly over several months between the U.S. Army (TRADOC System Manager, CATT and PM, CATT) and contractor Subject Matter Experts (SMEs). This effort resulted in the current CIS format and development process. As a result, CIS development is an approved process that provides a doctrinally correct combat behavior base for the S/W engineers to code. These CISs will also form the basis for maintenance activity on the delivered software. The CIS development process consists of the following steps.

Analyze the ARTEP task. The process of developing a BLUFOR[1] CIS begins with an analysis of a specific ARTEP task, e.g., 17–237–10–MTP, Tank Platoon, Collective Task: *React to Indirect Fires*. First a review of all doctrinal and training references pertaining to this task is made. This includes information in the ARTEP, pertinent tables of organization and equipment (TOE), Field Manuals, Training Circulars, Service School Training Texts, and other doctrinal literature. Prior to researching information for a specific CIS, information is generated for the unit type associated with the CIS, e.g., Tank Platoon in the case of the *React to Indirect Fire* CIS. This information is put into an appendix that is then attached to all CISs associated with the given unit type. The appendix provides common parameters for the unit, including capabilities and limitations, organization, platforms and weapons systems, employment techniques, and MOVE/SHOOT/COMMUNICATE/SEARCH/OBSERVE parameters, and success/failure criteria defaults.

Develop the general description statement. Next, the CIS developer describes in a brief paragraph the purpose of the task and how it is to be executed. This paragraph is informational for the software engineers so that they can get the "big picture" with respect to the CIS.

Specify initial conditions. The CIS developer then determines initial conditions and input data. Initial conditions describe the conditions, state, or posture of the unit at the commencement of the CIS, e.g., moving in a formation, stationary, etc.

Input data includes specific data from external sources that transform the generic CIS into an exercise–specific one. These data also must be known at the commencement of the CIS, and include the unit's direction, speed of movement, movement technique, center of mass, terrain limitations, known direction of threat, etc.

Describe subtasks and standards. The CIS developer then follows the ARTEPs subtasks and standards in exact sequence order and addresses each subtask/standard with doctrinal descriptions of how it is to be performed. Each performance "how to do" is described in terms of move, shoot, search/observe and communicate functions, as follows:

MOVE: Describe the sequence of steps that must occur to move the platforms or unit as associated with this particular subtask/standard. Describe movement directions, platform positioning and orientation, etc. Include parameters for speed, interval, etc.

SHOOT: Describe target priorities, fire control and distribution and engagement/termination conditions, as appropriate. Include effective fire and ammunition parameters, etc.

SEARCH/OBSERVE: Describe weapons orientation, search/observe sectors, and search techniques, as appropriate. Use degrees or clock method (the latter is preferred). Include search ranges for various types of targets, sector of observation, etc.

COMMUNICATE: Describe instructions to be disseminated or messages to be sent based upon control measures or other conditions/situation.

Describe situational interrupts. The developer then determines conditions or events that apply to the CISs that may cause immediate reaction by the unit or platforms to engagements by the enemy. These are called situational interrupts (SIs) and describe the conditions that cause the SAF unit to respond to enemy activities

[1] OPFOR CISs are developed in generally the same manner. The difference is that OPFOR CISs are not developed from ARTEPs (OPFOR Heavy doctrine does not have ARTEPs). They are instead developed by describing OPFOR Heavy collective tasks actions and behaviors in the order in which they occur. Additionally, the OPFOR CISs are not formatted into the MOVE/SHOOT/COMMUNICATE/SEARCH/OBSERVE parameters but rather are written in sequential order as behaviors take place over time. The parameters are included in the sequential description or are included in a separate appendix. This process provides the training unit with a doctrinally structured opposing force that can be analyzed and studied to determine its strengths and weaknesses.

or to selected terrain features. There may be as many as five such conditions: Contact with an enemy with/ without lethal weapons system; air attack; indirect fire; and certain terrain conditions (e.g., defiles, close terrain, obstacles that affect movement).

Describe termination conditions. Conditions are then described that cause the CIS to be terminated and the next event in the overall sequence of tactical activities to begin. Typically there are three such conditions: enemy contact is broken or a new more dangerous enemy contact is present, arrival at a control measure requiring change of formation, and a directed change of mission.

Describe battlefield operation systems (BOS). Finally, coordination requirements needed to satisfy each BOS area, if appropriate, are addressed. These are essential considerations needed to cause coordination and synchronization on the combined arms battlefield.

4.2 CIS Description

This section presents the sections contained in a typical CIS (sections shown in bold face type), and short descriptions of each section, where the title of the section is not self–explanatory (shown in italics). These sections are graphically depicted in Figure 3, Figure 4, Figure 5, Figure 6, and Figure 7.

4.3 CIS Excerpts

Figure 8 contains short excerpts from selected sections of an actual CIS – React to Indirect fire, as an illustration of the types of material contained in the CIS. Note that the individual sections are not complete, but only contain enough material to illustrate the *type* of material contained in each section.

5. CIS Software Analysis Description

The CIS software analysis process is responsible for translating the behavior descriptions contained in the English language CISs into detailed software requirements.

5.1 CIS Software Analysis Process

The CIS Software Analysis process provides a mechanism for insuring that the detailed software requirements are faithful to the intent of the natural language CISs. The input to the CIS software analysis activity is the completed and approved natural language CIS from the CIS development phase. The CIS software analysis activity consists of the following steps.

Create state transition diagrams and activity charts. State Transition Diagrams represent states, events, and activities. Conditions/guards are used when the event corresponds to checking a variable that is already available. An example would be an order that includes a variable that designates whether the platoon is working as part of a company or independently. Conditions are not used when the condition would involve a complex calculation. In this case, an activity is created that evaluates to the possible outcomes of the decision making process. This activity then produces a series of events such that each event represents one of the conditions in the State Transition Diagram.

COMBAT INSTRUCTION SET (CIS)
CCTT SEMI-AUTOMATED FORCES (SAF)

SECTION A. IDENTIFYING AND ADMINISTRATIVE DATA

1. CIS ID #: *The sequential identifying number of the CIS, using the letter B for BLUFOR and the letters HVY for OPFOR, followed by a four digit number. Blocks of CIS numbers are assigned to specific BLUFOR and OPFOR units.*
2. DATE PREPARED OR UPDATED:
3. CIS TITLE: *The name of the collective task or tactical behavior to be represented.*
4. TYPE UNIT: *The type and level of the unit, e.g., Tank Platoon.*
5. RELEVANT ENTITIES/PLATFORMS: *The entities that predominantly characterize the type and level of the unit. For example, for a BLUFOR tank platoon, this would be the M1A1/M1A2 MBT.*
6. NAME OF PREPARER: PHONE NO.:
7. APPROVING GOVERNMENT AGENCY: *The designation of the Government agency validating the CIS, e.g., TSM-CATT, Fort Knox KY, or Threat Support Division, Fort Leavenworth, KS..*
8. DATE APPROVED:
9. NAME OF APPROVING OFFICIAL:

Figure 3: CIS Section A – Identifying and Administrative Data

SECTION B. DOCTRINAL FRAMEWORK FOR CIS

1. **REFERENCES:** *Lists all doctrinal and training references from which information is extracted in developing the CIS.*

2. **ARTEP TASK AND NUMBER:** *Repeats the title of the collective task exactly as it appears in the ARTEP. For OPFOR that does not have ARTEPs, this paragraph identifies the source from which the task is drawn.*

3. **GENERAL DESCRIPTION OF TASK:** *Describes the purpose of the task and how it is to be executed.*

4. **ARTEP SUBTASKS AND STANDARDS:** *Lists in narrative form (BLUFOR only) an abbreviated version of the subtasks and standards of the collective task as they appear in the ARTEP. Subtasks and standards are not included for OPFOR, however, identification of other sources as appropriate, will be made.*

5. **INITIAL CONDITIONS:** *Includes the condition, state, or posture of the unit at the commencement of the CIS, e.g. , moving in line formation, expectation of enemy contact, etc.*

6. **INPUT DATA:** *Includes specific data from external sources that transfer the generic CIS into an exercise-specific one. These are the data that must be known at the commencement of the CIS, such as the unit's direction, speed of movement, movement technique, etc.*

7. **NOTES:** *Any explanatory notes that need to be included for clarity, such as reference to schematics, doctrinal definitions, etc.*

Figure 4: CIS Section B – Doctrinal Framework for CIS

SECTION C. ACTIONS TO BE TAKEN

1. **SEQUENCE OF ACTIONS:** *Describes the precise sequential actions that must occur in order to properly execute this tactical behavior. As a means of traceability, actions are described in the order in which the subtask and standard are delineated in the ARTEP. OPFOR actions are described as they typically occur.*

2. **TIME-DEPENDENT ACTIONS/RESULTS:** *Includes such things as time factors which may not be immediately apparent when "taking a snapshot" on the battlefield but which can have a cumulative impact over time. For example, "after 6 hours the battle position will have hasty mine-fields in place."*

Figure 5: CIS Section – Actions To Be Taken

SECTION D. CHANGES IN CIS STATUS

1. **SITUATIONAL INTERRUPTS:** *Indicates those selected conditions or events that cause immediate reaction by the unit to engagement by the enemy.*

2. **TERMINATING CONDITIONS:** *Describes the conditions that cause the CIS to be terminated and the next event in the overall sequence of tactical activities to begin.*

Figure 6: CIS Section – Changes in CIS Status

SECTION E. BATTLEFIELD OPERATING SYSTEMS (BOS) COORDINATION

BOS Coordination allows specific behaviors to be identified for BLUFOR units to reflect autonomous coordination. BOS is a BLUFOR term, for OPFOR this section is labeled COORDINATION. The BOSs are Maneuver, Fire Support, Air Defense, Command and Control, Intelligence, Mobility/Countermobility/Survivability, and Combat Service Support.

Figure 7: CIS Section – Battlefield Operating Systems (BOS) Coordination

```
    3.  GENERAL DESCRIPTION OF TASK:
        The indirect fire drill is used to minimize the effects of artil-
lery, mortar, or chemical attack.  If the platoon is on the move when
attacked by indirect fire, all vehicles maintain speed and direction
while moving out of the impact area.  If the platoon is stationary, tanks
move to covered, turret-down positions and continue the mission, or move
out of the impact area.  If the mission requires the platoon to remain
stationary, permission must be obtained from the company commander before
moving.  The PL initially sends a SPOTREP to the company commander, fol-
lowed by, situation permitting, a more detailed SHELREP.  (FM 17-15, p.
3-21; STP 17-19EK4-SM, p. 2-133)
    6.  INPUT DATA:
        a.  Specify the platoon's mobility condition (moving or station-
ary).
        b.  If moving, specify the platoon's formation, movement tech-
nique, direction of movement, and speed (use overlay, OPORD, FRAGO, ter-
rain reasoning).
        c.  If stationary, specify the platoon's center of mass.
SECTION C.  ACTIONS TO BE TAKEN
    1.  SEQUENCE OF ACTIONS:
        1.  Platoon is on the move and must react to indirect fires
[Condition #1]:
            a.  Immediately executes evasive action to avoid the impact
area (V)
                MOVE:  All vehicles execute evasive action without stop-
ping or changing general direction (i.e., increase to dash speed and
execute sharp left and right oblique turns).  (ARTEP 17-237-10-MTP, p.
A-15; FM 17-12-1, p. 12-101; IDT judgment)
```

Figure 8: CIS Excerpts

Generalize activities that span multiple echelon levels. Any activity that is used by more than one echelon level of the force structure is an excellent candidate for the common activities table (see below).

Add short descriptions for States, Information-flows, Events, Activities, and Controlling Activities that are not self explanatory. The short descriptions provide a high level description of the associated chart element. Short descriptions are contained in the data dictionary forms for the charts.

Complete the *Mapping to CIS* portion of the long descriptions. Long descriptions contain CIS Mapping statements, rationale (if any), and any notes or general comments that are too long to fit in the short description field for the chart element.

Submit the CIS State Transition Diagrams, CIS Activity Charts, CIS mapping in the long descriptions, and short descriptions for approval by the Subject Matter Experts (SMEs). This step forces both sides of the CIS analysis process to meet and reach agreement regarding the translation of the natural language CIS description into software requirements.

Review the individual activities in the CIS Activity Charts that are not already defined in the common activities table. The common activities table contains activities that apply across more than one CIS. Each of the "new" activities that are not in the list are reviewed to determine if the activity could possibly be used in another CIS. If it appears to be common, the level of commonality is determined. (For example, is it common to both OPFOR and BLUFOR platoons or just OPFOR platoons.) The potential common activities are denoted on the CIS Activity Charts and may later be added to the common activities table.

Review the CIS State Transition Diagrams and CIS Activity Charts with the software lead engineer. At this point the commonality level and names of the proposed common activities are approved for inclusion in the common activities table, if acceptable.

Complete long descriptions for the activities that are not currently denoted as common to more than one CIS. Long descriptions are required for activities and their associated events. The activities requiring long descriptions at this point include those which are unique to the given CIS, as well as those that were proposed for inclusion in the common activities table, but have not yet been included. In this step the long descriptions are also updated to reflect any assumptions or agreements that were reached in the reviews by the SME and the software lead engineer.

For all common activities (both newly designated as common and preexisting common activities), complete the fields *Is activity* and *Implemented by Module.* These fields point to the location of the module

which contains the definition of the activity along with the activity's long description.

Update the common activity list to include the new common activities. Once the new common activities have been determined, the responsible software lead engineer adds them to the common activities table.

For each newly defined common or unique activity, write one or more SRS requirements The software requirements for an activity are directly derivable from the long description for the activity.

Enter the long definition for each information–flow in the CIS Activity Charts that has not already been defined in the main model. For information–flows, the components of the flow, if there are any, are also entered. In this case, each of the components are defined as information–flows, including long descriptions, if they do not already exist. For any information–flow which has a range of values the values are specified in the long description. For example, an information–flow which corresponds to the role of the vehicle within a BLUFOR platoon can have the range of values from role one to role four.

5.2 CIS Software Analysis Products

This section provides examples of the products that are developed during the CIS Software Analysis activity. These products have all been generated using the Statemate CASE tool. These include state diagrams and associated activity charts, the data dictionary associated with the state and activity charts, the common activities table, a cross reference table that links the CIS standards to their associated activities, and the SRS as a final product. The examples are all derived from the CIS software analysis activity associated with the *React to Indirect Fire* CIS.

5.2.1 State Diagrams and Activity Charts

Figure 9 shows the state diagram for the B0013 platoon control operations. Each of the boxes in the diagram correspond to states that the platoon can be in while accomplishing the *React to Indirect Fire* CIS. The arcs in the figure represent transitions between states. The identifiers in capital letters (e.g. REQUEST_APPROVED) are *labels* for the transition, indicating the reason that the transition occurred. The conditions in square brackets ([ ]) represent *guard conditions* (i.e. conditions that must be met for the state transformation to take place). The format */st!* followed by an activity name in parentheses (e.g. /st!(PLT_GENERATE_REQUEST_TO_CDR)) indicates that the named activity is to commence at the specified point. The referenced activities are then shown with their associated data flows in the unit activity chart, Figure 10, which shows platoon activities to be accomplished while simulating the *React to Indirect Fire* CIS. The boxes in dotted lines represent external components that have communications interfaces with the activity.

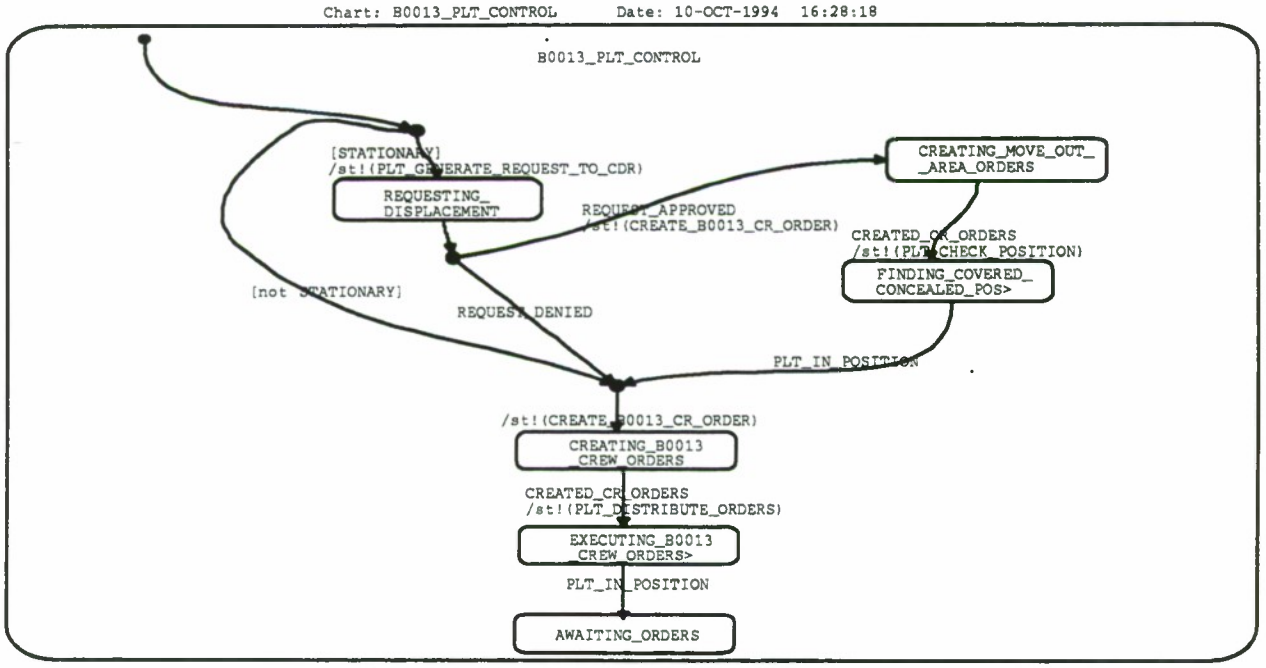


Figure 9: B0013 Platoon Control State Chart

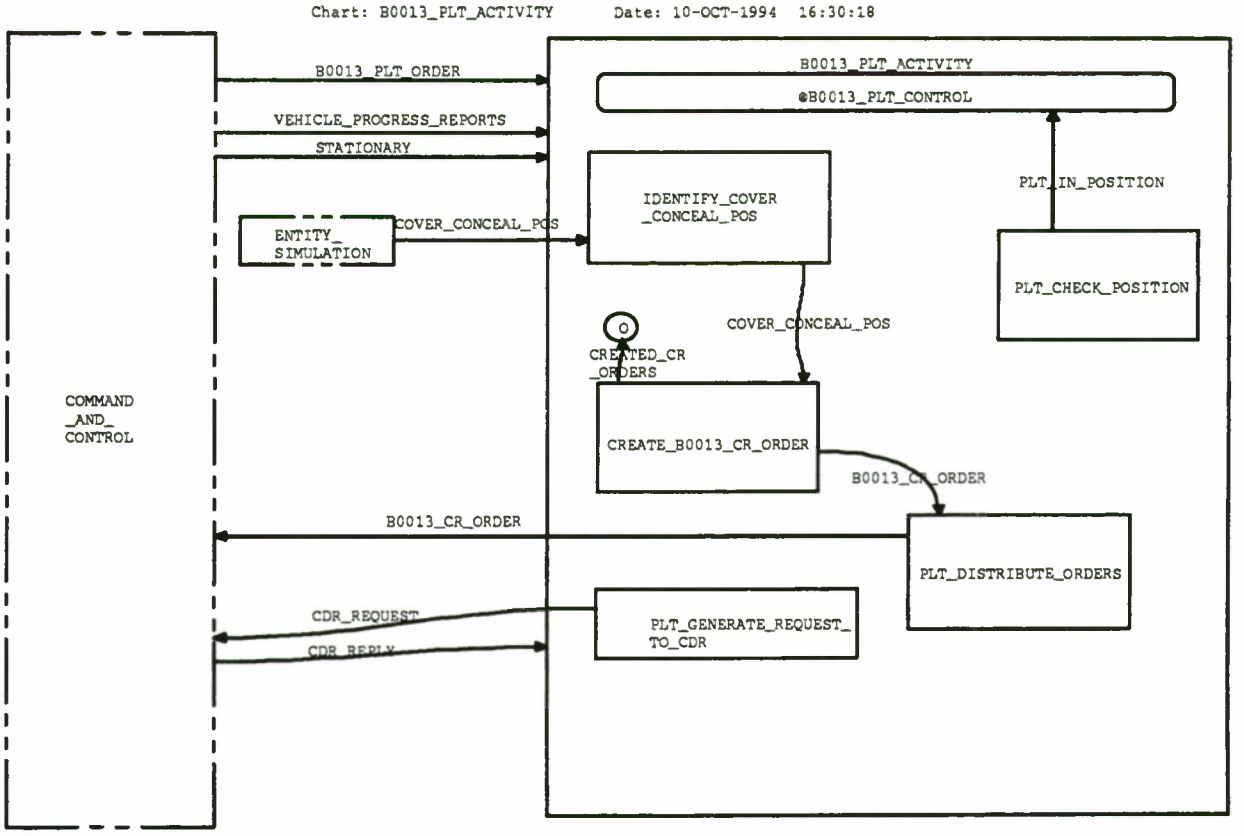


Figure 10: B0013 Platoon Activity Chart

As an example of the correspondence between the various charts, consider the reference to CREATE_B0013_CR_ORDER in Figure 9. This reference occurs in the transition to CREATING_B0013_CREW_ORDERS. This transition can come from several previous states, dependent on whether the unit was moving or stationary, or whether the unit was able to find a covered and concealed position. The actions specified by the CREATE_B0013_CR_ORDER activity are to be accomplished while the unit is in the state CREATING_B0013_CREW_ORDERS. The CREATE_B0013_CR_ORDER activity is then specified on the B0013 platoon activity chart, Figure 10In this chart, CREATE_B0013_CR_ORDER is shown providing a data output to the PLT_DISTRIBUTE_ORDERS activity, namely the B0013 crew order. It has a data input from the IDENTIFY_COVER_CONCEAL_POS activity, the covered and concealed location. It provides a logical output to the B0013 platoon control chart, the CREATED_CR_ORDERS flag, indicating that the crew orders have been completed. All of the activity descriptions for this activity are given in the data dictionary entry associated with the activity (see Section 5.2.2). The software requirements associated with the CREATE_B0013_CR_ORDER activity are then derived from the combination of its reference in the B0013 state transition diagram, its relationship with the other entries in the B0013 activity chart, and its description in the data dictionary.

5.2.2 The Data Dictionary

The data dictionary entries provide backup information pertaining to the state and activity charts presented in the previous section. In particular, the *notes* section documents any interpretation of the CIS that may have been required during the CIS Software Analysis activity. An example of a data dictionary entry from the *React to Indirect Fires* state and activity charts is given in Figure 11.

Activity CREATE_B0013_CR_ORDER
Defined in chart: B0013_PLT_ACTIVITY
Type: BASIC
Long description:
CIS STMT: C.1.1 & 2
DESCRIPTION: Crew Order to Perform BLUFOR Tank Platoon React to Indirect Fire is created using the inputs listed below.
INPUTS:
Mobility condition<HR>
Formation <HR>
Movement technique <HR>
Direction <HR>
Speed <HR>
Route <HR>
Center of mass
OUTPUTS:
CREATE_B0013_CR_ORDER
RATIONALE: None.
NOTES:
#2 – If platoon is stationary (input) and movement request is denied platoon stays in position (turret down positions). Any tanks not in turret down positions move to nearest turret down positions. For tanks staying in position the position given will its current position. Possible CISs using this condition: B0034 ,Execute Platoon Defensive Mission.
#3 – If platoon is stationary (input) and movement request is approved platoon moves 600m out of impact area and then moves into covered and concealed positions, taking evasive action and maintaining wingman orientation. Platoon then goes into awaiting orders. Possible CISs using this condition: B0008 ,Execute Herringbone Formation.
#1 – If platoon is on the move platoon moves out of impact area without changing direction using evasive action (at dash speed and execute sharp left and right oblique turns). Platoon moves at least 500m from initial point of impact.

Figure 11: Example from Data Dictionary

5.2.3 CIS–to–Activity Cross Reference Table

The CIS–to–Activity Cross Reference Table provides the link between the CIS standards and the activities derived during the CIS software analysis process. Figure 12 shows an example of a cross reference table created for the React to Indirect Fire CIS.

B0013 CIS Statement Cross–Reference Report

Statement	Activity/Rationale
C.1.1	CREATE_B0013_CR_ORDER
C.1.2	CREATE_B0013_CR_ORDER
C.1.2.c.1	PLT_GENERATE_REQUEST_TO_CDR
C.1.2.c.1	SHELREPs are not implemented in SAF. But the information will be included in Spot reports which are implemented as a general background routine and not shown in the charts.
C.1.2.c.2	PLT_GENERATE_REQUEST_TO_CDR
C.1.2.c.2	PL specifies a new formation, reassembles the platoon, and continues mission. These actions are represented by the state awaiting orders.
C.1.2.c.3	IDENTIFY_C_C_POS
C.1.2.c.3	PLT_GENERATE_REQUEST_TO_CDR

Figure 12: Example of Cross Reference Table

5.2.4 Common Activities Table

This section contains an excerpt from the Common Activities Table that pertains to the execution of the *React to Indirect Fires* CIS. Common activities are activities that are performed across units, echelons, or forces which are alike in their use and functionality. The Common Activities Table sample is shown in Figure 13.

Activity Name	Description
IDENTIFY_COVER_CONCEAL_POS	This activity finds a set of covered and concealed positions for an entity. The covered and concealed position will offer cover from entities that are approaching from a given direction. In addition, the position will be within a specified area and be able to conceal the entity. Inputs: Bounding Area (Rectangle) Enemy Location Entity Physical Characteristics Outputs: Covered and Concealed Position

Figure 13: Excerpt from Common Activities Table

The following list provides examples of various primitive state activities which have been developed out of the analysis:

- Adjust Spacing
- Cease Fire
- Determine Assembly Area Positions
- Determine Sectors of Fire
- Estimate Enemy Size
- Follow Route
- Generate Resource Report to Headquarters
- Move to Position
- Orient Weapon
- Scan Sector
- Vehicle Tether to Platoon

5.2.5 SRS Products

Figure 14 presents an excerpt from the SRS document directly related to the *React to Indirect Fire* CIS.

RCG93010: The Combat Unit Tactics capability shall request covered and concealed positions from the Environment capability when any of the following occur:
a. BLUFOR Tank Platoon executes a React To Indirect Fire Order
b. BLUFOR Tank Platoon executes an Occupy a Platoon Battle Position Order
c. BLUFOR Tank Platoon executes a Perform Platoon Fire and Movement Order
d. BLUFOR Mechanized Infantry Platoon executes a React to Indirect Fire Order
e. BLUFOR Mechanized Infantry Platoon executes a Hasty Dismount Order
f. BLUFOR Mechanized Infantry Platoon executes a Mount a Platform Order
g. BLUFOR Mechanized Infantry Platoon executes a React to Contact (Mounted) Order
h. OPFOR Motorized Rifle Platoon executes a Remount Platform Order
i. OPFOR Motorized Rifle Platoon executes an Occupy a Defensive Strong Point Order
j. OPFOR Motorized Rifle Platoon executes a Fire Engagement Order
k. OPFOR Motorized Rifle Platoon executes a Withdraw/Disengage Order
l. OPFOR Motorized Rifle Platoon executes an Occupy a Temporary Defensive Position Order
m. OPFOR ATGM Squad executes an Occupy a Defensive Position Order
n. OPFOR Motorized Rifle Company executes a Fire Engagement Order
o. OPFOR Motorized Rifle Company executes an Assault an Enemy Position Order.System Component: SAF
Analysis Activity: Identify_Cover_Conceal_Pos

Figure 14: Excerpt from SRS Document

6. Conclusions

The techniques described in this paper have been successfully used to describe and implement combat behaviors for the CCTT SAF program. The CIS description of behaviors has proven to be useful not only on the CCTT SAF program, but should also be of benefit to any DIS simulation program that requires a description of appropriate tactical behaviors. Using formal CIS software analysis techniques helped to bridge the gap between Subject Matter Experts, who know a great deal about tactical doctrine but very little about software development, and software engineers, for whom just the reverse is true.

7. References

[1] Defense System Software Development. *DOD–Standard–2167A.* February 29, 1988.

[2] CCTT Software Development Plan, Report Number 94–CCTT–LFS–00005. May 27, 1994.

[3] Turning Combat Instruction Sets into Software. CCTT Design Document, January 27, 1995.

[4] McEnany, B.R. and Marshall, H. CCTT SAF Functional Analysis. *Fourth Conference on Computer Generated Forces and Behavioral Representation.* May 1994.

[5] Ourston, D. and Bimson, K. Integrating Heterogeneous Knowledge Processes in the SAF Behaviors Implementation. *Fourth Conference on Computer Generated Forces and Behavioral Representation.* May 1994.

8. Author's Biographies

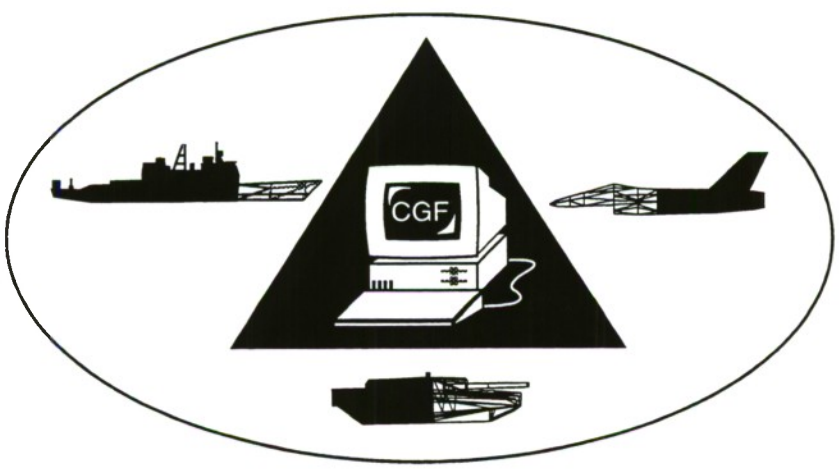
Dirk Ourston was responsible for the Behaviors software development activity during the early phases of the CCTT SAF program. He is currently working on internal research projects that seek to extend the reasoning capabilities associated with SAF command entities. He has a PhD in computer science from the University of Texas at Austin, with a specialization in artificial intelligence. He has over 25 years of experience in software design and development.

Ed Chandler is the on–site Subject Matter Expert (SME) for SAF. He has developed, written/reviewed extensive natural language CISs. Currently Mr. Chandler is reviewing CIS analyses of those CISs to be included in SAF. He has 30 years experience in the Army's joint tactical operation planning and execution, and has commanded from platoon through Brigade.

Elsie Loh is currently responsible for the CCTT SAF Behaviors CIS Analysis effort. She has a MSE in Computer Engineering from the University of Central Florida. She has 15 years experience in software design and development.

Dave Blanchard is currently responsible for the Behaviors software component of the SAF. He has a BS in computer science from the Virginia Polytechnic Institute and State University. He has 8 years experience in the design and implementation of Ada software.

Henry Marshall is the government lead on the CCTT SAF Concurrent Engineering team. Most of his duties have been in software and CGF acquisition support for STRICOM and NTSC. He received a BSE in Electrical Engineering and an MS in Systems Simulation from the University of Central Florida.


CGF

Implementation Of A Tactical Order Generator For Computer Generated Forces

David R. Pratt, Howard Mohn, Robert McGhee
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
pratt@cs.nps.navy.mil

1. Abstract

The Modular Semi-Automated Forces (ModSAF)[1] system has shown itself to be a valuable tool to generate forces in the synthetic battlefield (Calder et. al. 1993) (Loral 1993). Major projects, such as the Synthetic Theater of War (STOW) call for it use to simulate a significant number of entities. While it has proven quite capable of this, its modeling of the order generation process is some what lacking. Focusing exclusively on the third paragraph, Execution, of the Army's five paragraph order. The purpose of this work is to develop an interface that would more closely model the generation of the five paragraph order. A key component of this was the generation of both a ModSAF scenario and a printed five paragraph order.

2. Introduction

The creation of the scenarios form a five paragraph order required modifications to the current mission planning and task assignment process. Primarily, this involved the use of the Rational Behavior Model (RBM) to generate the task frames (Byrnes et. al. 1993). RBM was developed primarily as a means of mission planning and control for autonomous robots. Extending this concept to address the problems of mission planning for computer generated forces allows the human greater flexibility and capability in controlling large numbers of computer generated forces in a large-scale virtual environment.

A prototype mission planner was added as a library. Using the US Army's five paragraph operations order as the basis, modifications and enhancements were made to the standard Modsaf GUI editors to allow the user to generate the five paragraph orders. The editors provide information to the framework about which artificial intelligence modules operate on the data input from the order, generating ModSAF tasks that are subsequently executed by the company. Currently, the input is parsed directly into a series of company-level ModSAF mission tasks.

1. This paper assumes a basic knowledge of and familiarly with ModSAF. If the reader is unfamiliar with the system, please refer to the ModSAF User's Guide for an overview.

3. The Five Paragraph Order

The US Army's five-paragraph operations order (OPORD) is a standardized document that enables a trained reader to rapidly develop an understanding of the overall situation, mission, commander's intent for the operation, and tasks of subordinate units. This format is universally understood throughout the US Army, and thus is an intuitive way for a military user to assign orders to subordinate units. The five paragraphs are Situation, Mission, Execution, Service Support, and Command and Signal. Figure 1 contains a sample OPORD generated by the Tactical Order Generator (Army FM 10105-1) (ARMY FM 71-1).

Map overlays containing maneuver graphics are essential accessories to the basic text order. Overlays serve to graphically illustrate the contents of the order. In most cases, the OPORD text will refer the reader to these overlays, especially within the Situation and Execution paragraphs.

4. Initial Attempt: Distributed Design Strategy

The initial effort was focused on the development of a stand-alone application that would allow the use of a truly object-oriented language, such as C++. This language, unlike K&R C, supports stronger type-checking, polymorphism, and object inheritance. This was initially selected to make the application development easier, as C++ supports stronger type checking, and the use of objects for mission selection would closely follow the object-oriented paradigms in the Common Lisp Object System (CLOS) and CLIPS (Giarratano 1993) (Steele 1990) (Stroustrup 91).

A number of lessons were learned in pursuing this approach. The selection of a particular language depends

Operations Order for A
PO Database Number 1/1/359
Paragraph 1: Situation
a. Enemy forces.
Total Number of Enemy Combat Systems in Area of Interest: 83

b. Friendly Forces.
Total number of Friendly Combat Systems in Area of Interest: 60
Number of Tanks: 14
Number of IFV's: 4
Number of Other Vehicles: 2

c. Area of Interest (Company Level).
Southwest Corner Location: UTM Grid: 10SFQ546773
Southeast Corner Point: UTM Grid: 10SFQ595771
Northwest Corner Point: UTM Grid: 10SFQ542795
Northeast Corner Point: UTM Grid: 10SFQ593795

Paragraph 2: Mission
TF 6-40 AR/Berlin Bde attacks 45 minutes from now, from point UTM Grid: 10SFQ562784 to point UTM Grid: 10SFQ545788

Paragraph 3: Execution
a. Concept of the Operation
This Mission will be executed in 4 phases.
b. Detailed Instructions -- A Company:
Phase 1:
Attack along axis UTM Grid: 10SFQ551782
Assault from attack position, location UTM Grid: 10SFQ547785 to seize objective, UTM Grid: 10SFQ545788
Phase 2:
Transition from phase 1 to phase 2: on order.
Defend battle position UTM Grid: 10SFQ545788
Oriented on the TRP located UTM Grid: 10SFQ538792
Left Limit: UTM Grid: 10SFQ533789
Right Limit: UTM Grid: 10SFQ542795

Phase 3:
Transition from phase 2 to phase 3: on order.
Conduct a road march, Start Point UTM Grid: 10SFQ562784
End at Release Point UTM Grid: 10SFQ573805
Phase 4:
Transition from phase 3 to phase 4: continue.
Occupy Assembly Area at location UTM Grid: 10SFQ584805

Paragraph 4: Service Support
a. Supply
Supplies on hand:
Ammunition basic load: 100.00.
Fuel basic load: 100.00.
Resupply Points:
Ammunition Resupply Point Location: UTM Grid: 10SFQ582780
Fuel Resupply Point Location: UTM Grid: 10SFQ580784
b. Services
Battalion Aid Station Location: UTM Grid: 10SFQ584784
Admin Log Operations Center Location: UTM Grid: 10SFQ580778

Paragraph 5: Command & Signal
a. Signal.
Current CEOI in Effect.
b. Command.
Chain of Command is
Commander
Third Platoon Leader
First Platoon Leader
Second Platoon Leader

Figure 1. Sample Five Paragraph Operations Order Generated by the OPORD System

significantly on the language of the preexisting code. Unless one is willing to do a complete rewrite of the program, the programming language should be the same as the majority of the code that will be reused in the new application. A minimalist approach to code modification and extensibility should be pursued whenever possible. In attempting to use the large body of preexisting ModSAF code, changes were made that were inconsistent with good programming practices. The integration of the new code with the old code was not well-defined, and thus allowed inconsistencies in the application's execution.

5. Second Attempt: Integrated Design Strategy

The second attempt was much more successful, and involved the building of a separate library and incorporating it into the existing ModSAF code. This new library -- "LibOpord" -- was created and integrated in the same manner as the other subordinate ModSAF libraries. Analysis of the code structure for both versions of ModSAF revealed that the unit operations editor was the best choice for the insertion of the code to initialize and call LibOpord. This is a base editor defined in the LibUnits library that allows the user to enter a set of tasks for the selected unit, its subordinate units (if any), and individual vehicles. It links these tasks together through operator defined phase transitions, called enabling tasks. The unit operations editor was chosen as it is the one that appears when a unit or a vehicle is selected from the Plan View Display. As a result, a minimal change to one existing ModSAF library was required, in addition to the inclusion of the header file in the main.c preprocessor directives.

5.1 Integration of the Mission Planner Into ModSAF

The integration of LibOpord into the ModSAF library set was done in accordance with (Loral 1993). The library requires the following modifications to LibUnits to become available to the user:

- Modification of the data structure within LibUnits to include a Motif pushbutton widget that will call the LibOpord editor, add a pointer to the LibOpord data structure, and define LibOpord as an additional sub-editor.
- Inclusion of the initialization function within the LibUnits initialization routine ("units_create_editor(...)) that will allocate memory for the data structures and build the Motif GUI widget tree.
- Addition of a callback

(units_operations_order(...)) within LibUnits that handles the pushbutton mouse event.

Initialization of LibOpord is done as part of the LibUnits initialization steps; no other library requires modification. This form of library initialization and utilization is identical to the way other ModSAF editors are created and called.

5.2 Graphical User Interface Development

The base operations order editor was intended to be built using the LibEditor functions, but this proved unfeasible due to the irregular nature and complexity of the editor. Instead, the editor was created using a Motif widget tree that allows the programmer to build a customized GUI (Figure 2). The root of the widget tree is attached to the ModSAF base GUI, forming a branch that is displayed when called.

The subordinate editors were developed using the LibEditor library. The LibEditor create function is called in the LibOpord initialization function for each subordinate editor. Currently, there are nine subordinate editors that are initialized in this manner. Every subordinate editor has two corresponding functions that are called during run-time when the editor is displayed. The first function hides the base editor and calls a LibEditor function to display the selected editor. The second function collects the user input data when the editor is exited and control returns to the base editor.

5.3 Implementation Limitations

The OPORD editors constrain the user to a limited set of choices, which is significantly different than a free-text OPORD. There are several obvious reasons for this:

a. Natural Language Processing Limitations

The limitations inherent in natural language processing do not allow for rapid integration of the data input to the other modules in the mission planner. This problem is a subject of ongoing research; such a data entry system would be too complex and cumbersome to implement here.

b. Mission Simplification

One of the goals of the mission planner is to simplify mission determination and selection by the human. An extremely rich OPORD editor would only serve to complicate the generation of company level missions. Instead, a robust expert system should be able to compensate for the simplicity of input by reasoning about the circumstances of the input data and making decisions in the context of the assigned mission.

5.4 Data Formatter

The purpose of the Data Formatter is to ensure the artificial intelligence submodules of the Mission Selector/Evaluator receive the user input data in a usable

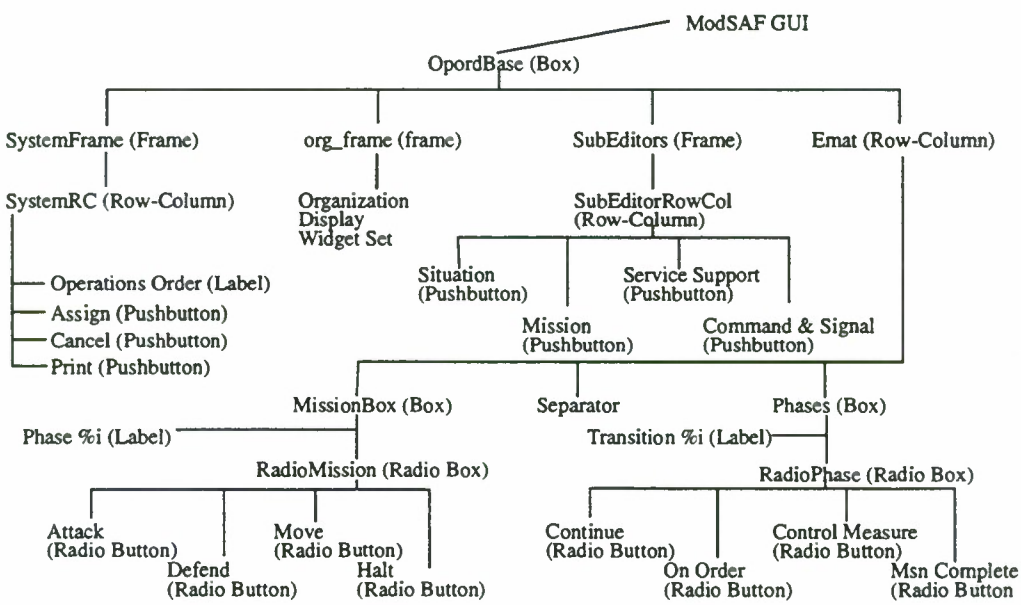


Figure 2. OPORD Widget Tree

form. The Data Formatter is a set of data structures and the code that converts the data from one structure to the other. Its current implementation is as a structure of structures. There is currently no modification being done as the artificial intelligence submodules are not developed. The user interface through LibEditor requires that the data displayed and entered must be placed in a separate structure for later processing. To simplify this procedure, a base structure is defined that contains the five paragraphs in separate structures (Figure 3). This compartmentalization of data allows one portion of the structure to be modified based on the user's selection.

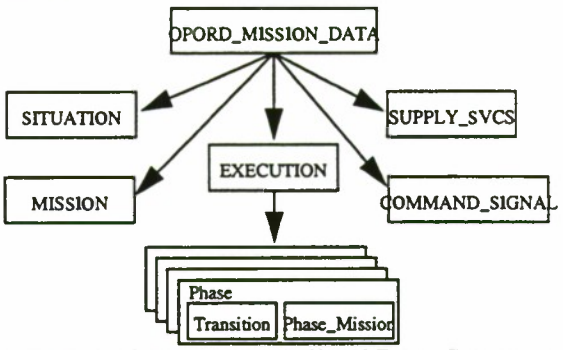


Figure 3. Compartmentalized Data Structure

The OPORD_MISSION_DATA structure aggregates the information from all editors into one structure. This approach provides the capability to easily change the data parameters in one centralized structure. This area is ripe for additional enhancements; the main danger here is overwhelming the user with data. The focus here is to request the essential data (keep it simple) and let the AI reason about the context and situation, and modify those parameters as required. Currently, the maximum number of phases for a given operations order is four. This limitation was a design choice. Most battalion-level operations orders never exceed four or five phases; this number can be changed through adjusting the value of the array variable through the constant MAX_OPORD_PHASES.

5.5 Mission Selector/Evaluator

This module was not implemented due to the time constraints involved. The shell about which the Mission Selector/Evaluator operates was successfully completed, but the initial attempts to develop a distributed mission planner consumed the available time. This submodule could be the subject of future work.

5.6 ModSAF Orders Generator

The ModSAF Orders Generator is also a prototype module. It makes extensive use of the new LibTaskUtil library, which is designed to allow direct creation of specified task frames without calling the task's associated editor. This allows libraries like LibOpord to generate a set of task frames and assign them to a unit, without human intervention. The library was modified by J. E. Smith to allow multiple task frames to be linked by enabling tasks. These modifications will become generally available in the next version update of ModSAF (Version 1.3). A single reader file was modified to include company-level tasks; the associated editors and libraries were passed to the taskutil_init function during initiation of the operations order structures and editors.

6. Summary

The distributed design strategy was more involved, but was probably due to faulty methodology than the strategy itself. The integrated design strategy resulted in the rapid development of a proof-of-concept prototype. While this strategy is the simpler of the two, it may be rendered unusable due to the potential resource requirements of the artificial intelligence modules. The prototype terrain reasoner, written in CLIPS, requires sixty seconds to determine a single route using A* search on a three kilometer by three kilometer terrain. Expanding this to cover an "average" battalion area of interest of five by five kilometers could easily triple the time required. Additionally, this would require heavy use of system resources, which may not be available due to the demands of ModSAF. Combining this submodule with other expert system submodules may make the integrated design strategy unfeasible, however, its advantages are the ability to rapidly develop and test a module within the framework of ModSAF.

The initial results from the prototype resulted in a significant simplification of task generation for the user. One operations order phase generated on the average two and a half ModSAF phases, with no requirements for additional parameter changes. Further research is needed, however, to fully determine the resource implications of including AI modules in an already complex system. The use of the operations order as a means to generate a company-level mission simplifies mission generation, but a robust expert system is needed to effectively convert the operations order input data to a set of ModSAF tasks.

7. Acknowledgments

The work described in this paper was funded by STRICOM. The work would not have been possible with out the help of Mr. Joshua Smith and Dr. Andrew Ceranowicz, both of LORAL-ADS.

8. References

US Army Field Manual 101-5-1, Operational Terms and Graphics, Headquarters, Department of the Army, October 1985.

United States Army, Field Manual 71-1, The Tank and Mechanized Infantry Battalion Task Force, Headquarters, Department of the Army, October 1988.

Byrnes, R. B., Nelson, M. L., Kwak, S.,McGhee, R. B., Healey, A. J. "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Underwater Vehicles," Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire, Durham, NH, September 27-29 1993, pp. 160-178.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., Ceranowicz, A. Z., "ModSAF Behavior Simulation and Control," Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, University of Central Florida, Orlando, FL, March 17-19 1993.

Giarratano, J. C., CLIPS User's Guide, CLIPS Version 6.0, Lyndon B. Johnson Space Center Information Systems Directorate, Software Technology Branch (NASA), 1993.

Loral ADS, "ModSAF Software Architecture Design and Overview Document," 1993.

Loral ADS, "A Modular Solution for Semi-Automated Forces -- ModSAF, An Overview," Loral ADS Briefing Slides, 1993.

Steele, G. L., Common LISP, 2d Ed., Digital Press, 1990.

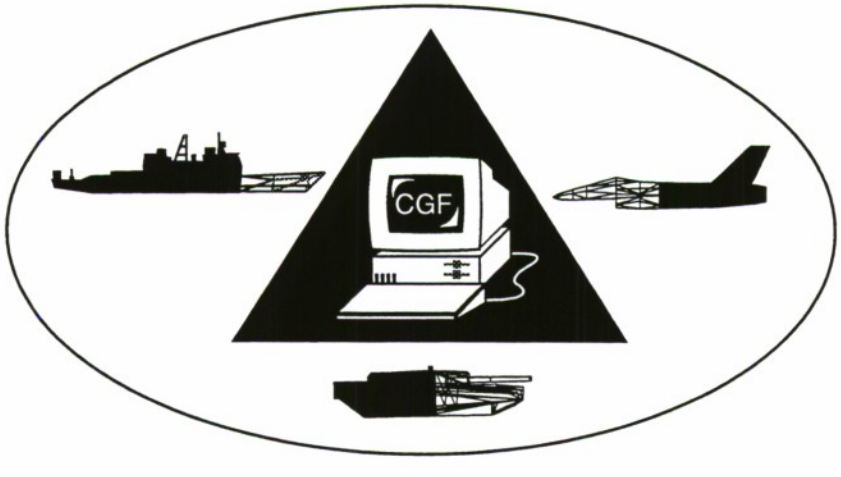
Stroustrup, B., The C++ Programming Language, 2d Ed., Addison-Wesley, 1991.

9. Authors' Biographies

Dr. David R. Pratt, an Assistant Professor in the Department of Computer Science at the Naval Postgraduate School. He is extremely active in the DIS community as both a developer and information resource for systems development and network integration.

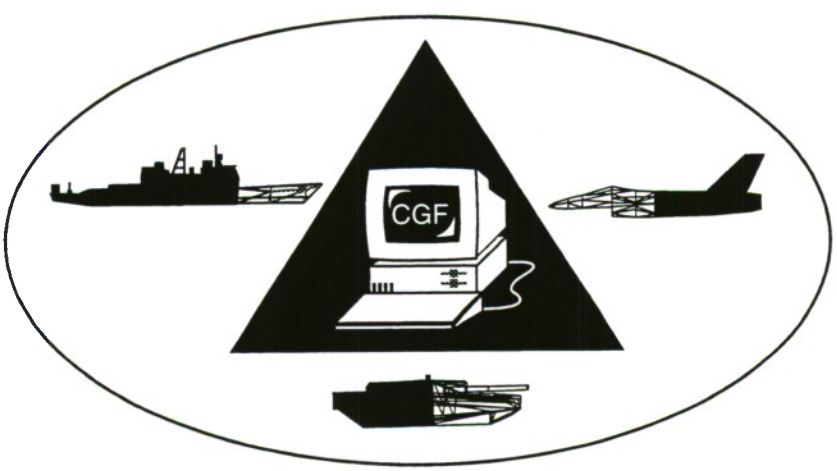
Maj. Howard Mohn, USA, earned his Masters in Computer Science from the Naval Postgraduate School in September, 1994. A former Armor Officer, he currently and Military Intelligence Officer serving in a bunker somewhere in Germany.

Dr. Robert B. McGhee is a full Professor in the Department of Computer Science of the Naval Postgraduate School. A Fellow of the IEEE, he lead the Ohio State Walking Machine project. His current interests include dynamics and autonomous entity interaction.


CGF

Session 6b: Route Planning I

Karr, UCF/IST
Longtin, Loral ADS
Schricker, UCF/IST


CGF

Unit Route Planning

Clark R. Karr and Sumeet Rajput
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
ckarr@ist.ucf.edu

1. Abstract

This paper presents a new Unit Route Planning Algorithm (RPA) based on a novel abstraction called the Obstacle Segment Abstraction (OSA). The OSA is a space efficient representation of obstacles within a terrain database The Unit Route Planning Algorithm combines the OSA with two route planning approaches, regular grid and vertex graph. An efficient search algorithm, A*, is applied to the grid to determine the optimal route. Factors in addition to distance are incorporated into the route cost determination to introduce trafficability and cover and concealment in the evaluation of the potential routes. The RPA has several strengths. It is scalable. It considers distance, trafficability, and cover and concealment. It generates multiple routes and identifies chokepoints. Finally, it can be used to find and progressively refine lengthly, precise routes.

2. Route Planning

Route planning occurs at multiple levels within CGF systems. At the lowest level, routes for individual vehicles are prepared that allow vehicles to move from one point to another. Vehicle routes typically consist of a series of line/curve segments. These piecewise routes are represented as a list of points (called "route points") and the vehicle is expected to travel along a line/curve between route points. A companion paper (Karr 1995) describes a mechanism for avoiding moving obstacles while traversing a route.

At levels above the vehicle, routes are prepared for groups of vehicles, i.e. units (platoons, companies, battalions). As the hierarchy of units is ascended, the unit's size and therefore the width of the route increases. Unit routes occur within "movement corridors" reflecting the fact that, formations and tactics aside, the vehicles' routes are only constrained to be within the corridor. A corridor has sufficient width for a unit to move through it. Planning routes for units of differing sizes may seem a simple matter of generating a route for a single vehicle and treating that route as the center of a corridor. While feasible, this approach retains the complexities of creating precise vehicle routes which increases the computational expense/time above that required for unit route planning.

This paper details research into route planning for military units. A novel approach based on the combination of two disparate route planning approaches is presented that:

1. is scaleable, suitable for battalion through vehicle route planning,
2. computationally fast,
3. considers distance, cover and concealment, and trafficability,
4. generates multiple acceptable routes between points within unit boundaries, and
5. finds and reports chokepoints within the routes.

This research has been done in conjunction with the development of a Computer Generated Forces (CGF) Automated Mission Planner capability. To generate and evaluate multiple courses of action to fulfill a mission, the Mission Planner requires multiple, tactically sound unit routes and the identification of chokepoints along the routes (Lee 1994).

2.1 General Path Planning Approaches

Motion planning with particular emphasis on robot path planning and robot manipulator path planning has seen considerable work, see Hwang et. al. (1992) for a survey. There are four broad categories of path planning approaches: free/blocked space analysis, vertex graphs analysis, potential fields, and grid (regular tessellation) based algorithms (Thorpe 1984). Each approach has strengths and weaknesses. See companion paper (Karr 1995) for a discussion of free/blocked space analysis and potential fields. The discussion of the vertex graph and regular grid approaches is repeated here.

In the vertex graph approach, the endpoints, vertices, of possible path segments are represented

(Mitchell 1988). This approach is suitable for spaces that have sufficient obstacles to determine the endpoints; determining the vertices in "open" terrain is difficult. In addition, representing only path vertices creates three other difficulties. First, trafficability over the path segments is not represented; route segments between arbitrary vertices are typically "open" or "blocked". Second, factors other than distance can not be included in evaluating possible routes. In the military simulation domain, concealment and cover are important factors in route planning. Third, because the width of route segments is not represented one of two problems occurs. Chokepoints (narrow sections of routes) are marked "blocked" or "open". If "blocked", acceptable routes are discarded. If "open", unacceptably long, narrow routes are accepted. Thus, the vertex graph approach has difficulty representing route width.

In the regular grid approach, a grid overlays the terrain, terrain features are abstracted into the grid, and the grid rather than the terrain is analyzed. Each grid cell is typically marked as "open" or "blocked". Quadtrees are an example of the regular grid approach (Mitchell 1988). Grid routes are converted into terrain routes typically by adding the z-coordinate to the xy-coordinates in the grid route. This approach simplifies the analysis but has two disadvantages. First, "jagged" paths are produced because movement out of a grid cell is restricted to four (or eight) directions corresponding to the four neighboring cells (eight to allow diagonal moves). Second, the granularity (size of the grid cells) determines the smallest "opening" that can be identified. If the granularity is too large, small openings in obstacles (e.g. bridges over rivers) are lost. A small granularity is required to capture small openings which increases the computational expense of the analysis.

3. Unit Route Planning

The Route Planning Algorithm (RPA) presented here combines a unique obstacle abstraction with two route planning approaches, regular grid and vertex graph. A regular grid overlays the terrain. The size and location of the grid is determined by the unit boundaries of the unit planning the route. The scale of the grid is determined by the unit size; grid scales less than the typical "frontage" of the unit give good results. For example, grid scales from 75 to 125 meters are suitable for platoons, 250 to 500 meters for companies, and 750 to 1000 meters for battalions. Obstacles on the terrain are encoded in the grid using a novel abstraction called the Obstacle Segment Abstraction (OSA). The A* search algorithm is applied to the grid to determine the optimal route. Cost factors for distance, trafficability, and cover and concealment are included in the evaluation of the potential routes.

This work was performed in the IST CGF Testbed, an environment for testing CGF behavioral control algorithms developed under the sponsorship of ARPA and STRICOM (Danisas et. al. 1990, Gonzalez et. al. 1990, Petty 1992, Smith et. al. 1992a, and Smith et. al. 1992b).

3.1 Terrain Grid

CGF systems operating within DIS type environments rely on a representation of terrain termed a Terrain Database (TDB). This research was performed using the SIMNET TDB which encodes terrain and terrain features as polygons; these are encoded in turn in edge and vertex lists. This TDB contains features such as treelines, canopies, rivers, and lakes. These features are the physical obstacles. The RPA is applicable to any polygonal TDB format (e.g., ModSAF's CTDB (Smith 1994)) and to TDB formats that represents obstacles as distinct features.

When a route is requested, a regular grid is laid over a portion of the terrain. The location and size of the grid are determined by the unit boundaries and the orientation of the grid is determined by the orientation of the destination to the starting location. The granularity of the grid (the size of each grid cell) is determined by the unit size. Each grid cell has eight neighboring grid cells. The cells to the north (N), south (S), east (E), west (W) are the orthogonal cells and the cells to the northeast (NE), northwest (NW), southeast (SE), and southwest (SW) are the diagonal cells.

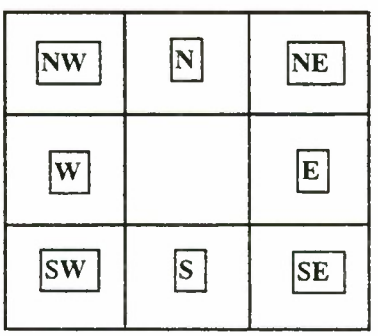


Figure 1 Neighboring cells

3.2 Obstacle Segment Abstraction

The terrain underlying the grid is analyzed and each obstacle in encoded in the grid. The encoding uses a small set of linear segments called the Obstacle Segments (OSs). The different types of OSs are: horizontal and vertical, diagonal, and tunnel as shown in Figure 2.

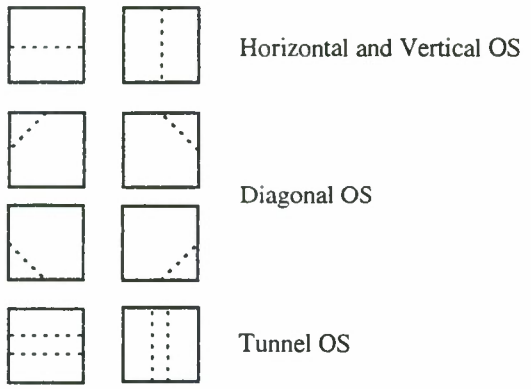


Figure 2 Obstacle Segments

Each OS is identified with an Obstacle Identification Number (OIN). The OSs representing a single terrain feature are assigned the same OIN. Thus, each OSA represents a single terrain feature and is the set of OSs with the same OIN.

Treating physical obstacles as OS abstractions is the basis for OSA route planning. The precise polygonal details of the obstacle are be dispensed with and an encoded representation is used in its place. As will be seen, the grid granularity determines the "correlation" error between the feature and its abstraction. So long as the granularity reflects the unit size, the correlation error is not a significant issue.

3.2.1 Creating Obstacle Segments

The intersections of terrain obstacles and the edges of the grid cells are determined and converted to OSs. An OS is created between the sides of the two entry points of an obstacle into a grid cell. An obstacle that does not exit a cell does not impose a barrier to travel within the cell; a vehicle can simply move around it.

Obstacles with width (e.g., rivers and lakes) will sometimes be represented by more than one OS in a grid cell. This occurs when the edges of the obstacle cross different grid cell boundaries. For example, Figure 3 shows some notional area of terrain and Figure 4 shows the resulting OSA grid.

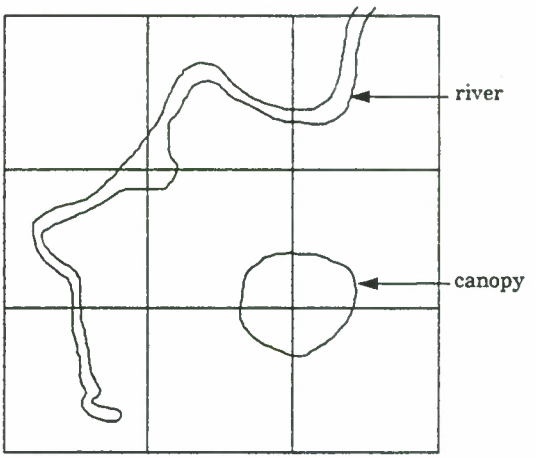


Figure 3 Notional Terrain

The OSs representing each obstacle are assigned unique OINs. In this example, the river's OSs are assigned the id "1" and the canopy's OSs are assigned the id "2". The river crossing the corner of a grid cell causes the river's OSA representation to include two cells with multiple OSs.

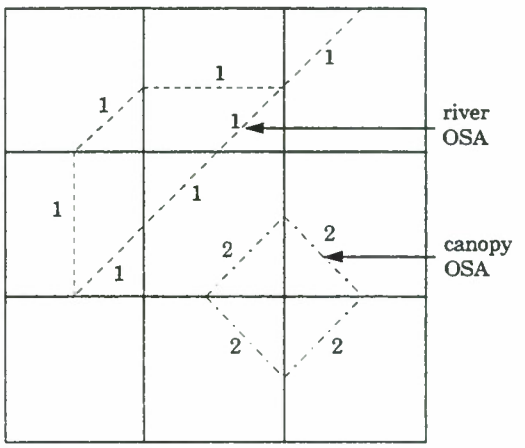


Figure 4 Obstacle Segment Abstraction Grid

3.3 Creating Routes

Route planning within the OSA grid is a matter of *searching* the grid for optimal routes. The search through the grid utilizes the vertex graph approach to route planning. Within each grid cell, "sample" points (see Section 4.3.2) are isolated. Beginning at the start location, a partial route to each

neighboring cells' sample points is created provided that the partial route does not cross an Obstacle Segment. In turn, each partial route is extended. This is a graph search problem. An efficient algorithm for performing graph searches is A*, Winston (1992).

The Unit Route Planner calculates route cost from distance, trafficability, and cover and concealment. As the length of a route increases, its cost also increases. Route segments that are not concealed from enemy positions increase total route cost reflecting increased "danger" along those route segments. Poor trafficability similarly increases the cost of route segments. For this work, terrain slope and soil type governed trafficability. Flat terrain was the least costly. Soil types were given costs in relation to the ease of travel over them.

3.4 Advantages of the OSA Approach

The OSA approach combines the strengths of the vertex graph and the regular grid approaches and solves or alleviates their problems. Consider the shortcomings of the regular grid approach. First, the OSA encoding of obstacles within the grid is more sophisticated than the typical "open" or "blocked" approach. This encoding allows the interiors of cells containing obstacles to be considered for route segments. This removes the restriction that grid granularity is dictated by the smallest opening in the terrain. Second, within each grid cell, candidate route vertices called "sample points" are identified. The introduction of sample vertices into the regular grid solves the "jagged" path problem of regular grids. Consider the shortcomings of the vertex graphs approach. First, determining vertices in open terrain is not a problem because the grid overlays open terrain and vertices, sample points, are found within all grid cells. Second, trafficability and cover and concealment are encoded in the grid cell and are included in the evaluation of candidate routes.

4. The Unit Route Planning Algorithm

This section discusses the Unit Route Planning Algorithm that was implemented in the IST CGF Testbed. Specifically, Section 4.1 presents the *PlanRoute* algorithm. Section 4.2 discusses how the grid granularity can be set to the optimum value. Finally, Section 4.3 presents details of the algorithm implementation.

4.1 Algorithm PlanRoute

Algorithm *PlanRoute* plans a route between two points.

Algorithm PlanRoute

Input:
The grid to route on, the start, and destination points.

Output:
A list of points defining the route.

Variables:
route_list: A list of routes, sorted in ascending order of estimated total route cost. The first route on this list is referred to as $route_list_1$.
route_to_be_extended: The least costly (and hence the first) route on the *route_list*. It is the same as $route_list_1$.
reachable_points: A list of points that are reachable from the last point on *route_to_be_extended*.

1. [Create the grid]
2. [Fill the grid]
3. [Initialize route_list]
   route_list = empty
4. [Are we there?]
   If the first route on the route_list terminates at the destination then
   4.1 Return $route_list_1$
   else
   4.2 [Extend least costly partial route]
   4.2.1 *route_to_be_extended* = $route_list_1$.
   4.2.2 Determine the points reachable from the last point on *route_to_be_extended* (described in Section 4.3.6). *reachable_points* = $\{r_1, r_2, \ldots, r_n\}$.
   4.2.3 Expand *route_to_be_extended* to each of the points in *reachable_points* if another, less expensive route to those points does not exist.
   4.2.4 Add these new routes to *route_list* in ascending order of route cost and underestimate.
   4.2.5 Go to step 4.

Steps 1 and 2 in the algorithm create the grid and populate it with abstract obstacles (see Section

4.3.1 for details on converting polygonal features to OSs). Step 3 initializes the *route_list* to empty. Step 4 finds the route.

In step 4, a check is made to determine if *route_to_be_extended* has reached the destination. If so, the result is sent to the caller; otherwise, *route_to_be_extended* is extended.

Step 4.2 extends the least costly partial route. The first route on the *route_list*, $route_list_1$, is removed and assigned to *route_to_be_extended*. All the points that are reachable from the end of this route are determined and stored in the list *reachable_points* (see Section 4.3.6 to see how reachable points are determined.) The route is extended to each *reachable point* only if another, less costly route does not exist. These "extended" routes are added to the *route_list* in sorted order and step 4 is repeated.

4.2 Grid granularity auto-detection

It is easy to see that at certain grid sizes the density of obstacles on portions of terrain can exceed the capacity of the OSA grid to represent them. When the granularity of the grid is too coarse for the density of obstacles, the Unit Route Planner attempts to create too many identical OSs. The Unit Route Planner detects when this occurs and prevents analysis from continuing at the same granularity. There are at least two solutions to the problem of a too coarse grid granularity. First, the algorithm can simply stop and replan the entire route at a finer granularity. This is the simplest solution conceptually. The algorithm simply "halves" the grid width and starts over. The second solution is similar to the quadtree representation. In this approach only the granularity of the grid cell that is too coarse is increased. This approach avoids the computational expense of repeated replanning but is more complex. For this work, the first approach was chosen.

4.3 Algorithm details

Sections 4.3.1 through 4.3.7 describe aspects of the PlanRoute in greater detail.

4.3.1 Marking Obstacle Segment Abstractions in Grid Cells

Physical obstacles (treelines, canopies, rivers, and lakes) that cross a cell's boundaries are marked as Obstacle Segments for each grid cell. In the polyonal SIMNET TDB, obstacle edge and polygon lists within terrain patches were searched and converted to OSAs.

4.3.2 Sample points

In the typical regular grid approach to route planning, the center of the grid cell is the single available route point. The Unit Route Planner has instead a set of 12 available route points called sample points. The sample points are the vertices used by A* in searching the OSA grid for routes. Figure 5 shows the sample points in relation to the diagonal OSs, a horizontal OS, and a vertical tunnel OS. They are arranged so there is at least one sample point on each side of each OS.

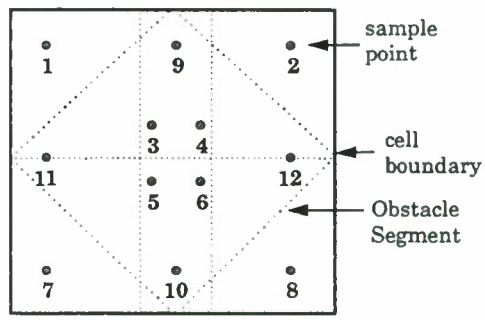


Figure 5 Sample Points

All grid cells have at least the first eight sample points. If the grid cell contains a tunnel, four additional points (9..12) are added.

Regular grids typically show a "digitization bias", Mitchell (1988), in which only 4 (8 if diagonal moves are allowed) angles out of each cell are available; these angles correspond to moves to the orthogonal and diagonal cells. This digitization bias causes the "jagged" appearance of routes. Sample points greatly reduce digitization bias. Each cell has between 512 angles and 1152 angles. Each partial route can be extended to between 64 and 96 available sample points.

4.3.3 Adjusting start and destination points

Because of the correlation error between an obstacle and its OSs, the start and destination points can not be translated to the OSA grid directly from their locations on the terrain. Simply put, the start or destination points must remain on the correct "side" of the Obstacle Segments. Figure 6 illustrates the problem. The start point in the OSA grid must be to the southwest of the river's OS.

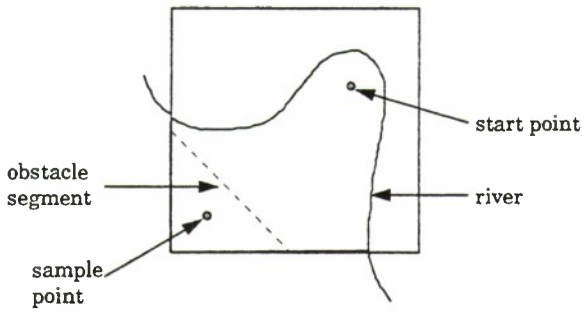


Figure 6 Start point adjustment

An algorithm was devised to move the start and destination points so that their positions relative to the abstract obstacles matched their positions relative to the physical obstacles. Specifically, the start or destination point is moved to a sample point (see Section 4.3.2) inside the grid cell. The algorithm is based on the *floodfilling* approach as described in Foley et.al. (1991). The edges that are "colored" by the floodfill are analyzed to determine which sample point to use as the terminal location.

4.3.4 Movement between grid cells

OSAs are approximations of the underlying terrain obstacles. OSAs in a cell and in orthogonal cells may "touch" on the edge between the cells even though the obstacles do not touch. This creates an artificial barrier to routing. To solve this problem the concepts of the "shared point" and "Obstacle Segment Displacement" were introduced.

4.3.4.1 Shared points

The point of contact of two OSs is called a *shared point*. Shared points are the midpoints of the edge separating two cells. Figure 9 illustrates a shared point.

4.3.4.2 Obstacle Segment Displacement

Obstacle Segment Abstractions may be aligned so as to block movement that would otherwise be possible if routes were being generated with respect to the underlying terrain features only. Consider, for example, the movement from cell A to cell B in Figure 7.

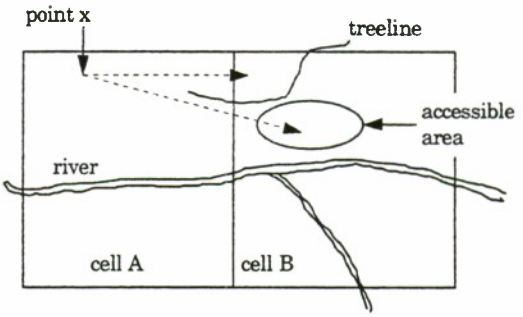


Figure 7 Access to region between physical obstacles in two cells

It is clear that a route may be extended from point **x** to the accessible area in cell B.

The OSAs for these two grid cells restrict access to the accessible area. Without adjustment, the only move from **x** is to the NW of the treeline:

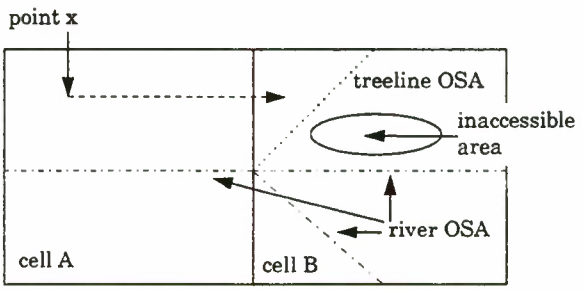


Figure 8 The corresponding obstacle abstractions

To prevent "touching" OSAs from unrealistically eliminating route segments, OSAs are "displaced" away from shared points. This is reasonable; the terrain features do not touch (they would be represented as the same OSA if they did), so the OSAs should not touch.

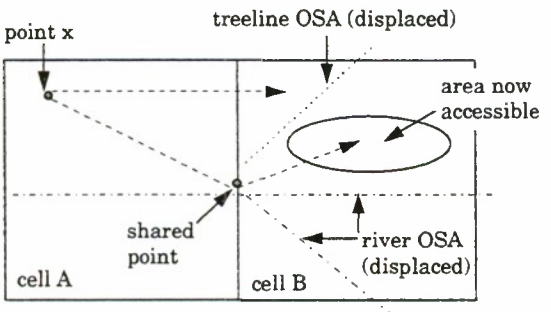


Figure 9 Displaced obstacle segments

In Figure 9, the OSA representing the river is displaced to the South and the treeline OSA is displaced to the North. The displacement opens

the gap between different OSAs making movement between them possible.

4.3.5 Extended Obstacle Segments

The *PlanRoute* algorithm (Section 4.1) considers only two grid cells at a time when extending partial routes. In Figure 10, a move is being considered from "current position" in cell A to y in cell C. The move appears to be valid when only cells A and C are considered; knowledge of OSs in cell B is not used. However, such moves should be disallowed because the OSA in cell C will extend into cell B.

To disallow such moves, "Extended OSs" consisting of three components, the Obstacle Segment and two extensions (one from each endpoint), are created. Each extension is a line segment at a right angle to its edge of the grid cell. When determining if a sample point can be reached, Extended OSs are checked.

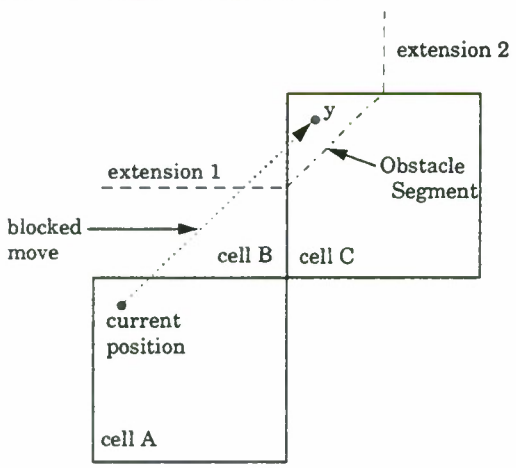


Figure10 Extended Obstacle Segment

4.3.6 Determining Reachable Points

Routes are extended to "reachable" points in neighboring grid cells. A sample point is reachable if a line segment to the point does not intersect an OS.

If two adjacent grid cells do not contain a shared point (Section 4.3.4.1), line segments are drawn from the end of the route to each of the sample points in the destination grid cell. If a line segment intersects an OS, the sample point is unreachable and discarded.

However, if two adjacent grid cells contain a shared point, the shared point is checked for reachability. If so, the sample points are checked for reachability from the shared point. Thus, a sample point is reachable if it is reachable from a reachable shared point.

Reachable points are partitioned into *mutually reachable sets* (MRS) which have the property that all points in a MRS are mutually reachable. Reachable points partitioning controls the combinatorial explosion of partial routes introduced by the multiple sample points. If any point in a MRS is involved in a route, all sample points in the MRS are considered to be part of the route and are not considered when extending additional partial routes. Thus, a MRS defines a reachable area and the least costly sample point within a MRS is used for routes into that area.

For example:

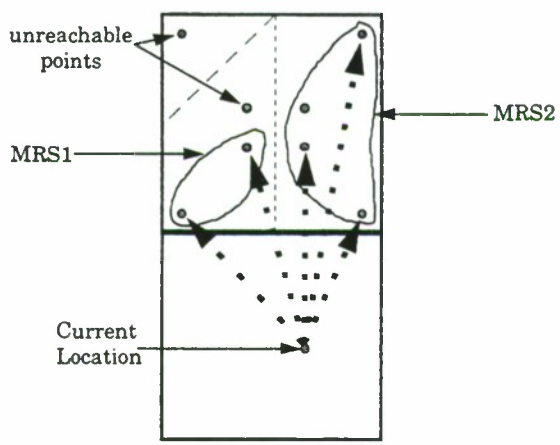


Figure 11 Mutually Reachable Sets

In Figure 11, sample points 1 and 3 in the North grid cell are unreachable from "Current Location". The reachable sample points have been partitioned into two MRSs corresponding to two possible destination areas in the North cell from "current location". Two partial routes will be created from Current Location into the North cell. The partial route into MRS1 will be to the least costly of MRS1's two sample points. The partial route into MRS2 will be to the least costly of MRS2's four sample points. Hence, two, rather than six, partial routes are created for further analysis by A*.

4.3.7 Computation of route cost

The A* algorithm evaluates partial routes on their "cost". The cost of a route, partial or whole, is the sum of the costs of its route segments and an underestimate of the remaining cost to the

destination. Three factors are considered in determining the segment cost: distance, trafficability, and concealment. Segment cost is the cost of moving between sample points in adjacent grid cells and is made up of three components:

- The *distance* between points.
- The *trafficability* which is calculated from:
  a. the *base cost* of the terrain in the destination grid cell and,
  b. the *slope* of the terrain around the destination sample point.
- The *percentage of intervisibility* from an area around the destination sample point to the areas around enemy locations.

The cost of a segment increases with length. Two factors governed trafficability: slope and type of terrain. Flat terrain had minimal cost with cost increasing with slope. The base cost is an average determined from the types of terrain in the grid cell. The average is calculated from 25 points taken from a regular 5 x 5 grid within the grid cell. For example, the base cost for a grid cell mostly on sand is greater than for a grid cell mostly on dirt.

There are other approaches to calculating the base cost. A simple approach would use only the terrain under the sample point. This approach ignores the terrain along the route. Another approach is to sample the terrain along the route segment. This approach considers each route to have zero width. Zero width routes have a major defect. Because unit routes have width (i.e. units travel along corridors), the terrain under the corridor is not introduced into the cost calculation. For example, narrow corridors are weighted the same as wide corridors. Although the approach taken in the RPA may consider too much terrain, it has the advantage that the terrain under a corridor is considered. Recall that grid granularity is determined by the size of the unit; so, the expectation is that most of the area under a grid cell will be traversed by the vehicles in the unit. Calculating an average over a grid cell appropriately increases the cost of grid cells with slow-go or no-go terrain.

Concealment is considered by calculating a percentage of intervisibility from a circular area around the destination sample point to the areas around enemy positions known by the routing unit.

Parameters into the algorithm control how much weight is given to trafficability and concealment. Thus, concealment can be weighted more to obtain routes that maximize concealment over trafficability. See Rajput (1994) for a thorough discussion of the cost calculation.

4.3.8 Multiple "Optimal" Routes and Chokepoints

This research was done in conjunction with the development of a CGF Automated Mission Planning capability. The Mission Planner requires multiple, tactically sound unit routes between points and the identification of chokepoints along the route.

The Unit Route Planner generates multiple "optimal" routes between two points. As each route is generated the terrain covered by the route is made "less desirable" for subsequent routes by increasing the base cost of grid cells under the route. Hence, the base cost of "used" grid cells increases and subsequent routes tend to avoid repeating the previous routes. Each route is "optimal" considering that previously generated routes should be avoided.

Keeping track of previous routes allows chokepoints to be identified. When multiple routes between points are requested, route segments corresponding to narrow corridors and chokepoints must be reused. The Unit Route Planner identifies multiple used route segments as chokepoints.

5. Conclusions and Future Work

The Unit Route Planner is flexible and efficient. Its flexibility comes from its scalability. The granularity (i.e. size) of the underlying grid varies based on the size of the routing unit. The finer the granularity, the closer the Obstacle Segment Abstractions correspond to the physical obstacles. Fine granularity routing is suitable for finding precise vehicle routes while coarser granularity routing is suitable for finding unit routes. The smaller the unit, the finer the required granularity. The Unit Route Planner algorithm can be applied to other polygonal TDBs.

The efficiency of the Unit Route Planner derives from three factors. First, the Obstacle Segment Abstraction approach allows obstacles to be represented with sufficient precision for routing but not so precise as to waste computational power. Second, the scaleable grid approach allows the representation of the terrain to correspond to the

requirements dictated by the size of the unit. These two factors together prevent unnecessarily precise (i.e. computationally expensive) routes from being generated. Third, the Obstacle Segment Abstraction in combination with the vertex graph approach allows the application of an efficient search technique, A*, to the problem of determining routes.

The Unit Route Planner has additional strengths. First, three routing factors, distance, trafficability, and concealment, are considered in finding optimal routes. The relative contribution of each routing factor is controlled by parameters to the algorithm. Hence, optimal routes of different characteristics can be determined. For example, concealment can be weighted more heavily than distance to produce predominately concealed but lengthy routes. Second, the Unit Route Planner generates multiple "optimal" routes between two points. As each route is generated the terrain covered by the route is considered less desirable. Subsequent routes tend to avoid the previous routes. Hence, each route is "optimal" considering that previously generated routes should be avoided. Third, chokepoints are identified. When multiple routes between points are requested, route segments corresponding to narrow corridors and chokepoints must be reused. The Unit Route Planner identifies those route segments as chokepoints. Fourth, narrow corridors between "close" obstacles are found. That is, corridors narrower than the granularity of the grid are represented. Fifth, the Unit Route Planner can be used to find and refine lengthy, precise routes through a process of successive refinement of routes. That is, a coarse route generated with a coarse grid granularity can be refined by applying the Route Planning process to sections of the coarse route at successively finer granularities. This approach would provide an efficient mechanism for planning detailed, lengthy routes. Sixth, although the goal of this research was a scalable, unit route planner, the Unit Route Planner is also an efficient vehicle route planner. It requires only the addition of a route smoothing algorithm to plan realistic vehicle routes.

In the current work, only terrain obstacles that are uncrossable (canopies, rivers, and treelines) are abstracted into OSAs. There are other features in the terrain which may prevent or hinder a unit's movement across them. These features are considered "no-go" and "slow-go" areas. For example, extremely steep terrain is a no-go area for many units. No-go areas could easily be represented as OSAs which would further decrease the combinatorial explosion of partial routes within the Unit Route Planner. Sandy and swampy terrain are slow-go areas because vehicles cannot move quickly over such terrain. An interesting area for future research would be to extend the OSA concept to representing slow-go obstacles.

One mechanism for representing slow-go obstacles is to add a "cost" for crossing each obstacle. In the current implementation, obstacles are considered to have infinite cost; this prevents movement across them. Slow-go areas could be represented with OSAs that have finite costs for crossing them.

Finally, it seems possible to extend the OSA approach from regular tesselation (the OSA grid) to irregular tesselation. Polygonal TDBs typically represent the ground surface with an irregular tesselation; e.g., as a set of triangles. It seems likely that the OSA approach could be implemented directly on these polygonal representations with minor modifications.

6. Acknowledgement

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

7. References

Danisas, K., Smith, S. H., and Wood, D. D. (1990). "Sequencer/Executive for Modular Simulator Design", *Technical Report IST-TR-90-1*, Institute for Simulation and Training, University of Central Florida, 16 pages.

Foley, J. D., van Dam, Andries, Feiner, S. K., Hughes, J. F. (1991). "*Compute Graphics Principles And Practice*", 2nd edition, Addison-Wesley Publishing Company, Inc.

Gonzalez, G., Mullally, D. E., Smith, S. H., Vanzant-Hodge, A. F., Watkins, J. E., and Wood, D. D. (1990). "A Testbed for Automated Entity Generation in Distributed Interactive Simulation", *Technical Report IST-TR-90-15*, Institute for Simulation and Training, University of Central Florida, 37 pages.

Hwang, Y. K. and Ahuja, N. (1992). "Gross Motion Planning--A Survey", *ACM Computing Surveys*, Vol. 24, No. 3, pp.219-291.

Karr, C. R., Craft, M. A., and Cisneros, J. E. (1995). "Dynamic Obstacle Avoidance",

Proceeding of the 5th Conference of Computer Generated Forces and Behavorial Representation, Orlando, FL, May 9-11, 1995, to be published.

Lee, J. J. and Fishwick, P. A. (1994), "Simulation-Based Planning for Computer Generated Forces", *Proceeding of the 4th Conference of Computer Generated Forces and Behavorial Representation*, Orlando, FL, May 4-6, 1994, pp. 451-459.

Mitchell, J. S. B. (1988). "An Algorithmic Approach to Some Problems in Terrain Navigation", *Artificial Intelligence*, Vol. 37, pp. 171-201.

Petty, M. D. (1992). "Computer Generated Forces in Battlefield Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 56-71.

Rajput, S. and Karr, C. R. (1994). "Unit Route Planning" *Technical Report IST-TR-94-42*, Institute for Simulation and Training, University of Central Florida.

Smith, J. E. (1994), *ModSAF Programer's Guide: LibCTDB*, Loral Advanced Distributed Simulation, Cambridge, Massachusetts.

Smith, S. H., Karr, C. R., Petty, M. D., Franceschini R. W., and Watkins, J. E. (1992a). "The IST Computer Generated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, University of Central Florida.

Smith, S. H., and Petty, M. D. (1992b). "Controlling Autonomous Behavior in Real-Time Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 27-40.

Thorpe, C. E. (1984). "Path Relaxation: Path Planning for a Mobile Robot", *CMU-RI-TR-84-5*, Carnegie-Mellon University The Robotics Institute Technical Report, April 1984.

Winston, Henry Patrick (1992). *Artificial Intelligence*, Third Edition, Addision-Wesley, 1992.

8. Authors' Biographies

Clark R. Karr is the Computer Generated Forces Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Sumeet Rajput is an Associate Engineer in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Concealed Routes in ModSAF

Michael J. Longtin
Dalila Megherbi
Loral Advanced Distributed Simulation
50 Moulton Street, Cambridge, MA 02138

1. Abstract

The ModSAF CGF system currently has the capability of attempting to find routes that are concealed as much as possible from the enemy. Two fundamental problems must be overcome in order to do this. Firstly, given information about the enemy, the system must compute the areas that are concealed or may be concealed from the enemy. This involves analyzing terrain features and performing intervisibility calculations. Secondly, given a list of concealed regions, a route from a specified starting point to a specified goal point which is the most favorable in terms of concealment must be found. This paper explores both problems, gives an overview of how they are currently implemented in ModSAF, and suggests enhancements that might be made in order to improve the current algorithm.

2. Introduction

The need to find concealed routes with respect to enemy locations arises frequently in CGF systems, since there are many real-life tactics which require this ability, such as bounding overwatch maneuvers or reconnaissance missions. A concealed-route algorithm is a vital component of CGF systems since it is a major contributor toward the realism of the generated forces. Such is the case with the ModSAF (Modular Semi-Automated Forces) CGF system sponsored by ARPA and STRICOM. A new concealed-route algorithm has recently been developed and incorporated into ModSAF. This algorithm consists of two independent components: the concealment-map generator and the concealed-route planner. A fundamental element of the concealed routes solution approach presented in this paper is the utilization and the exploitation of graph representation and search techniques.

The concealment-map generator uses terrain reasoning to compute concealed areas. It takes information about the enemy as an input and produces a series of concealed-area polygons, referred to as the "concealment map", as an output. The enemy information is specified is a list of "enemy descriptors". An enemy descriptor can be a direction (when it is known that the enemy is in a certain direction), an area (when it is known that the enemies are located or are maneuvering in a particular area), or a location (when the actual location of an enemy is known). The contribution of each enemy descriptor to the concealment map is computed separately, and the final concealment map is computed by taking the intersection of these.

The concealed-route planner takes a concealment map, a source point, and a goal point as inputs and produces a series of route points as an output. In particular, a graph is constructed where each concealed region produced by the concealment-map generator is treated as a node. A generalized technique is developed to automatically and efficiently determine which pairs of nodes (concealed regions) need to be linked together. In particular, to improve the problem-solving performance, the generalized technique uses some additional information about the problem at hand to decide which node successor should be expanded next (instead of blindly expanding all possible node successors) and which nodes should be disregarded (pruned from the search graph), while still heading and progressing toward the goal point. Finally, a concealed route from the source point to the goal point is found by applying one of the graph search techniques. The planner uses the A* search technique to find an optimal path through the concealment-map space. In this context, an optimal path is one which has the smallest total exposed distance.

The concealed-route algorithm uses the CTDB (compact terrain database) representation of terrain to ob-

tain terrain elevation and feature information. The compact representation is favorable because more terrain can be stored in physical RAM, thus decreasing the access time for terrain data. The algorithm also uses a non-preemptive asynchronous ring-based scheduler to allow searches to be distributed over time while giving the rest of the simulation a chance to run.

This paper describes the approach, implementation details, and future enhancements for the concealment-map generator and the concealed-route planner.

3. General Approach

The problem of finding concealed routes is not a trivial one. The algorithm must be flexible enough to accommodate information about the enemy with varying degrees of specificity; in a real-life battle situation, information about the enemy may be available in many different forms. An actual enemy location may be known, it could be known that several enemy vehicles are moving about in a given area, or as little as the general direction of the enemy could be known.

The algorithm must also have a keen sense of terrain awareness, including information about terrain features and the terrain skin. The ability to perform intervisibility calculations must also be available.

Thirdly, a concealed-route algorithm must be able to integrate information about the enemy and terrain into a route which is concealed from the enemy as much as possible.

The approach used in the concealed-route algorithm currently employed by ModSAF divides the problem into two independent sub-problems. The first is that of finding areas of the terrain which are concealed from the enemy, and the second is that of finding a route which best utilizes these concealed regions.

The first sub-problem is handled by the concealment-map generator, which accepts information about the enemy and performs an analysis of the terrain to compute which areas are concealed. The second is handled by the concealed-route planner, which accepts a series of concealed regions and produces the route with the best concealment. The concealment-map generator and the concealed-route planner will now be described in detail.

4. The Concealment-Map Generator

The purpose of the concealment-map generator is to accept information about the enemy and transform it into a series of areas which are likely to be hidden from the enemy. The concealed areas are described by a series of polygons which collectively comprise a significant portion of what is known as the *concealment map*. The concealment map contains information about the search area, the enemy, and the concealed areas.

4.1 Approach

There are two major areas of difficulty that need to be addressed in order to successfully implement the concealment-map generator, the first of those being information representation and the second being information processing.

The two types of information that must be represented are the terrain information and the enemy information. Fortunately, the former is handled by ModSAF's `libctdb` (Compact Terrain DataBase), which provides an interface for terrain feature extraction and the capability to perform intervisibility calculations. The latter problem is solved by *enemy descriptors*, which will be described later.

Once the terrain and enemy information is stored and available for processing, the concealed regions must be computed. This is explained next.

4.2 The Algorithm

The user of the concealment-map generator may specify an arbitrary number of *enemy descriptors*. An enemy descriptor is a piece of information about the enemy situation and can be one of the following: a direction, a location, or an area. The contribution of each enemy descriptor to the concealment map is computed separately, and the final concealment map is the intersection of these.

The processing of an enemy descriptor starts with the production of a grid of values. Each value in the grid corresponds to a location in the area that is being searched for concealed regions. This area is called the *search area* or *concealment-map space*. If a grid location has a value of one, then that location is concealed from the enemy described by that particular descriptor. If it has a value of zero, it is exposed.

The grid portion of the concealment map due to the first enemy descriptor is computed and copied verbatim to a cumulative concealment map. Next, the grid portion of the concealment map due to the second enemy descriptor is computed, and the intersection of

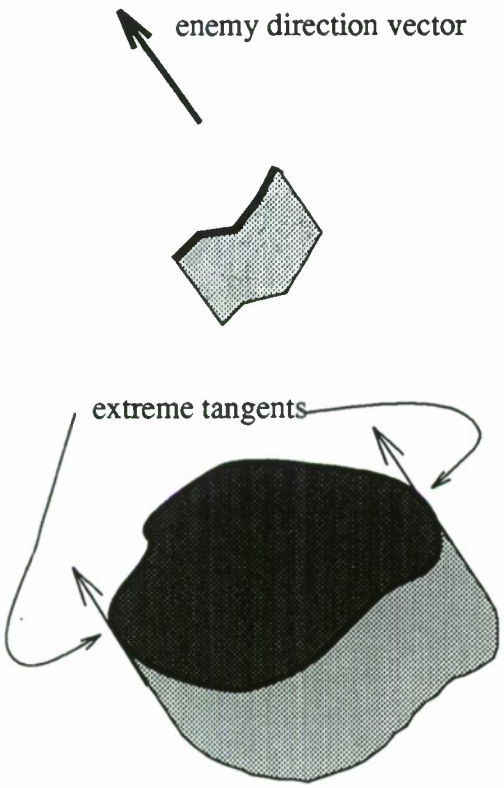


Figure 1: Concealed Areas Behind Features

that concealment map and the cumulative concealment map is computed. The result of the intersection is copied into the cumulative map. The intersection of two concealment maps is simply the logical ANDs of the corresponding grid locations. The contributions of each of the enemy descriptors are "merged" with the cumulative concealment map until all of the enemy descriptors have been processed.

One beneficial effect that results from the cumulative concealment map is that the information pertaining to all of the enemy descriptors is contained in the concealment map, and any enemy descriptor can be added to the concealment map without losing the information from the previously-specified enemy descriptors. For example, consider that information about enemies A, B, and C are known at time t_0, and a concealment map is computed for these enemies. Later, at time t_1, a new enemy, D, is spotted. The concealment-map generator can be invoked, specifying only the enemy descriptor for enemy D, and the resulting concealment map will contain areas that are concealed from enemies A through D.

After the last enemy descriptor has been processed, the grid portion of the cumulative concealment map is polygonalized, that is, polygons which enclose the clusters of ones in the grid are computed. The polygonalization must be performed because the route planner needs a list of polygons as an input.

4.3 Enemy Descriptors

This section describes how the concealment map contributions of each type of enemy descriptor is computed.

4.3.1 Enemy Direction

An enemy direction is usually specified only if neither the area that the enemy is occupying nor the exact location(s) of the enemy is known. This descriptor contains the least amount of information among the three types of descriptors. It is represented by a two-dimensional vector.

The contribution of an enemy direction to the concealment map is computed by applying a heuristic-based (non-exact) technique and includes areas behind treelines, buildings, and tree canopies. No elevation data is taken into consideration since the elevation of the enemy is not known. The heuristic used is that areas behind objects with respect to the enemy direction are suitable for concealment. Note that trees and buildings are assumed to be tall enough to provide concealment.

The concealed areas are found by first performing a search for terrain features such as treelines, tree canopies, and buildings. The challenge lies in computing areas behind these features with respect to the enemy direction. For linear features (i.e. treelines), this is relatively easy. The "front" of the polygon is just a copy of the vertices of the linear feature. The "back" of the polygon is found by adding a vector to each of the vertices of the linear feature. This vector has a direction which is the negative of that specified in the enemy descriptor. The magnitude of the vector is simply the depth of the concealed region, or how far back the region extends from the linear feature. See Figure 1 for an example of a concealed region behind a linear feature.

Finding areas behind polygonal features like buildings and tree canopies is slightly more difficult. The challenge lies in determining the front of the concealed region, which is actually the back of the polygonal feature. In order to find this, the extreme tangents of the polygonal feature in the direction of the enemy vector must be found. This is done by first converting the vertices of the feature into a coordinate

system whose **y**-axis is aligned with the enemy direction vector. Then, the vertices with the minimum and maximum **x**-values are found. These vertices are the first and last points of the back of the feature, as well as those of the front of the feature. Once these points are found, the back has to be distinguished from the front. This is done by first choosing one of the tangent-point vertices and taking the dot product of two vectors: the vector from the tangent vertex to an adjacent vertex, and the enemy direction vector. If the dot product is negative, then the adjacent vertex is part of the back of the feature. See Figure 1 for an example of a concealed region behind a polygonal feature.

4.3.2 Enemy Location

An enemy location may be specified when the exact location of an enemy is known. Since more exact information about the enemy is known in this case, an exact (non-heuristic-based) method is used to generate the concealment map contribution of an enemy location.

The grid portion of the concealment map is generated simply by using `libctdb`'s intervisibility engine. The intervisibility from the enemy location to each grid location is computed. If intervisibility to a given grid location passes, a "zero" is assigned to its corresponding entry in the concealment map. Otherwise, a "one" is assigned, meaning that the point is concealed.

4.3.3 Enemy Area

An enemy area may be specified when it is known that the enemy are moving about in a certain area. It may also be specified if the exact location of a given enemy is not known, but the general area in which an enemy exists is known or assumed. The concealment map contribution of an enemy area is computed by using a combination of exact- and heuristic-based techniques.

The difficulty with finding concealment from an area stems from the fact that there is an infinite number of locations within an area. Since all of these locations cannot be processed, a reasonably small number of them must be selected. This is where the heuristics come into play. A number of *observation posts* from within the specified area are selected. Observation posts are locations within the enemy area from which the highest level of visibility to the concealment-map area is available. Each observation post is then treated as an enemy location. The contribution of each observation post is integrated into the concealment map. This provides a near-worst-case scenario for enemy visibility (i.e. the case where the enemy has the best possible visibility).

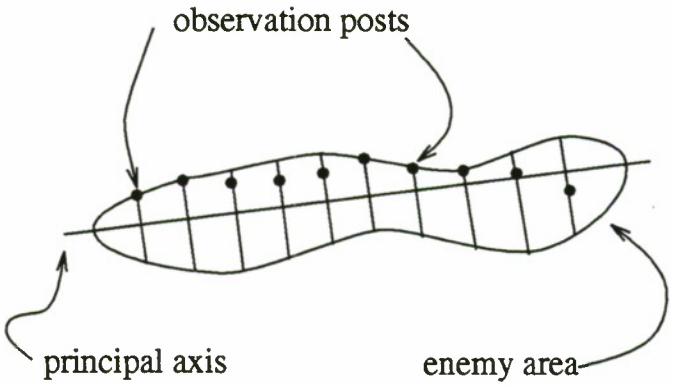


Figure 2: Observation Posts Within an Enemy Area

The algorithm which computes the locations of the observation posts is reasonably straightforward. First, the principal axis of the search area is computed. Next, a series of line segments which are perpendicular to this axis are computed. The distance between these lines depends on the grid spacing of the concealment map, and the line segments are bounded by the enemy area (i.e. their endpoints lie on the edge of the enemy-area polygon). The point along each line segment with the highest elevation is chosen as an observation post. Refer to Figure 2 for an illustration.

5. The Concealed-Route Planner

The purpose of the concealed-route planner is to generate a route which passes through concealed areas as much as possible while progressing toward a goal. The planner is given a start point, a goal point, and a list of concealed regions.

The route planner goes through three main phases during the course of a plan: a graph is constructed, the graph is searched for the most optimal route, and some post processing is performed on the route to doctor it. Each of these phases will be described in detail.

5.1 Constructing the Graph

The first phase of the concealed-route-planning algorithm is the graph construction. The graph consists of a series of nodes, and segments which interconnect the nodes. The nodes in this case are line segments that

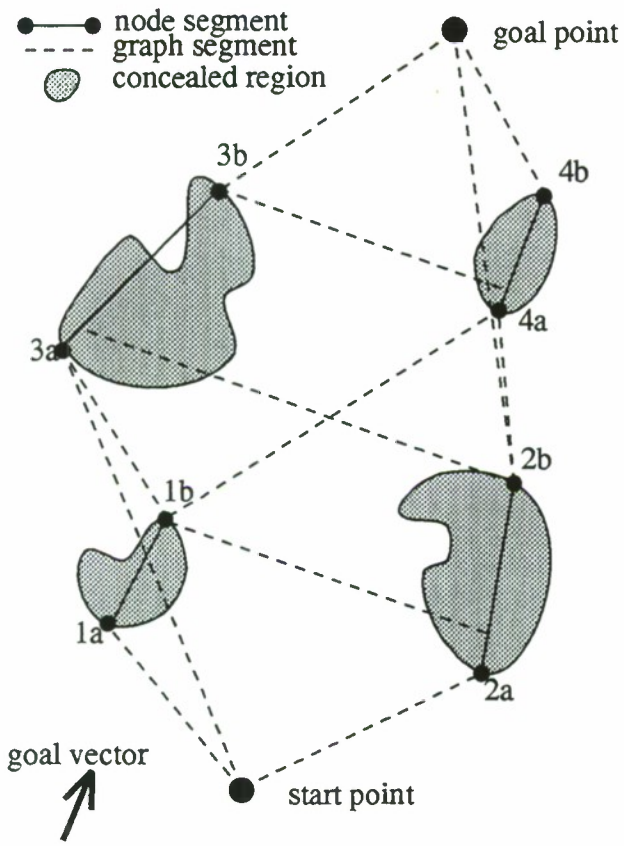


Figure 3: An Example of a Graph

are calculated directly from the concealed regions; each concealed-region polygon has a corresponding node in the graph. The line segments which represent the nodes will be referred to as *node segments*. The segments which interconnect the nodes will be referred to as *graph segments*. Note that the actual number of nodes exceeds the number of concealed-region polygons by two. These extra nodes are located at the start and goal points.

5.1.1 Calculating Node Segments

In order to calculate a node segment from a concealed-region polygon, the vertices of the polygon are first converted into a coordinate system whose **x**-axis is aligned with the *goal vector* (the vector from the start point to the goal point). See Figure 3. The vertex with the minimum **x**-coordinate and that with the maximum **x**-coordinate are found. These two points are the endpoints of the node segment for that concealed-region polygon. In effect, each concealed-region polygon is simplified down to a node segment. This greatly speeds up the search, since not all vertices of the polygons need to be processed during the search. Note that the node segments are not necessarily parallel to the goal vector. The direction of the node segment depends on the locations of the vertices of the concealed-region polygons.

5.1.2 Calculating Graph Segments

Two heuristic rules are applied in the construction of the graph segments. Firstly, no graph segment can go backwards, that is, the component of a graph segment in the direction of the goal vector must be positive. This ensures that the route never turns away from the goal. This rule is applied in order to eliminate excessively long routes.

The second heuristic rule concerns the selection of successor nodes. The maximum number of successors of a given node cannot exceed three, that is, the end of a node segment can be connected to at most three nodes. If there are more than three possible next nodes, the three closest are chosen. This rule came about with the assumption that a "next node" which is farther away than three other "next nodes" has a low probability of being chosen over the other nodes. Considering only the three closest nodes has the effect of eliminating lots of unnecessary searching, which improves the computational efficiency of the search.

Refer to Figure 3 for an example of a fully-constructed graph. Note that no graph segment points away from the goal and there are at most three successor nodes for any given node.

5.2 Searching the Graph

Once the graph has been constructed, it is searched for an optimal path. The A* search algorithm is used to perform the search (Tanimoto 1987). In essence, the search algorithm considers all possible paths along the graph segments from the start point to the goal point and assigns a weight to each path. The path with the minimum weight is chosen as the optimal path. The mapping from a path to a weight is done by an *optimization function*. The choice of optimization function is extremely important, and greatly impacts which route is chosen.

The definition of an "optimal route" is derived from a set of criteria which is determined before the search begins. The optimization function is derived directly from this set of criteria. In this case, the optimal route is the one with the lowest total exposed distance. The total length of the route which does not traverse concealed areas (i.e. the length of the graph segments along the path) should be minimized and is

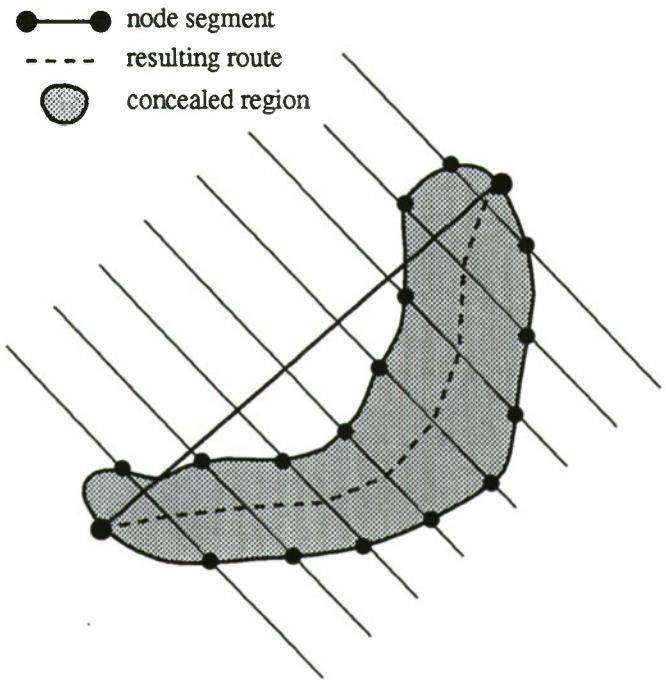


Figure 4: The Route Post-Processing Phase

used as the optimization function.

During the search, paths with weights greater than the euclidean distance from the start point to the goal point are eliminated from the search, since these paths can not possibly be the most optimal. This speed-enhancing technique is known as *pruning* (Barr et. al. 1981).

The search algorithm produces a list of nodes through which the optimal route passes. The actual route is then constructed from the endpoints of the node segments. The most optimal route for the example in Figure 3 is probably the one which traverses nodes two and four.

5.3 Route Post-Processing

It is possible for a node segment to leave and re-enter a concealed region, since concealed-region polygons are not guaranteed to be convex. An example of this can be found in Figure 3. In this case, node segment number three leaves and re-enters its concealed region. Such segments are modified in a post-processing phase after the search has been completed.

This post processing involves replacing node segments which are not completely contained within concealed regions to route segments that are. This is done by first constructing a series of evenly-spaced line segments which are perpendicular to the node segment. For each line segment, the intersections that it makes with the concealed-region polygon are found, and the average of these points is assigned as a route point. The result is a route which follows the middle of the polygon. See Figure 4.

Note that it is possible, although highly unlikely, for there to be more than two intersections between one of the line segments and the concealed-region polygon. In this case, the polygon is not processed, and the original node segment is used.

6. Usage

The concealed-route functionality is contained in a library of terrain-reasoning routines in the ModSAF system (Longtin 1994). Its interface consists of the following four public functions:

```
tr_create_concealment_map()
tr_destroy_concealment_map()
tr_generate_concealment_map()
tr_plan_concealed_route()
```

The first of the above functions allocates memory for a concealment map and initializes it. The user supplies the actual search area and the spacing of the grid points. The search area and grid spacing information is kept in the concealment map structure.

The second function simply frees the memory associated with a concealment map.

The third function actually begins the concealment-map generation. The user specifies the concealment map for which to compute concealed regions, an array of enemy descriptors, and a search state. The search state is a structure which keeps track of information pertaining to the search in progress. This is needed because the search is distributed across multiple simulation ticks. `tr_generate_concealment_map` schedules a function which actually performs the search to be called periodically. When the search has finished, a `ready` flag in the search state is set. The application must therefore poll this flag after calling `tr_generate_concealment_map` and wait for it to be set before calling `tr_plan_concealed_route`.

Finally, the fourth of the above functions actually attempts to find an optimal route through the concealment-map space. The user specifies a concealment map, a start point, and a goal point. A

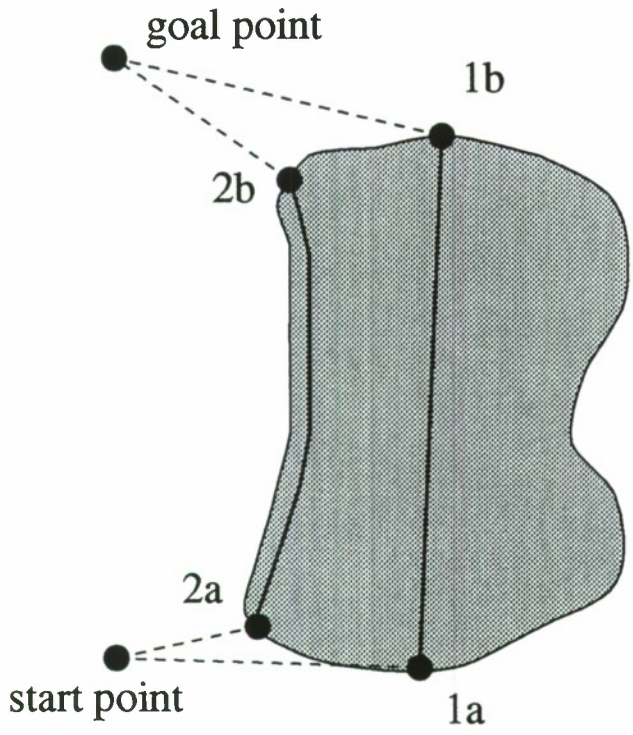


Figure 5: Large Concealed Regions

list of route points representing the concealed route is returned.

7. Future Enhancements

The current implementation of concealed routes in ModSAF is a first-cut attempt at producing a flexible concealed-route planner. This section describes future work that may be done to enhance the planner.

7.1 Node-Segment Generation

There are several problems with simplifying each concealed region down to a line segment. Firstly, as stated earlier, a node segment may leave the concealed region with which it is associated. This necessitates the inclusion of the post-processing phase of the route planner.

Secondly, the current design of node segments may lead to routes that are unnecessarily long. This may happen in cases where concealed regions are relatively large. Consider Figure 5. Obviously the current route planner will find a route which traverses the single node, passing through points 1a and 1b. The route goes through the middle of the concealed region, while a shorter route along the edge, passing through points 2a and 2b, would be more optimal because such a route is not only shorter, but the exposed segments of the route are shorter.

A third problem with the current method of node segment generation has to do with exposed segments. Currently, all graph segments are considered to be completely exposed, when in actuality, they may not be. This can lead to inaccuracies in computing the weights of the prospective routes. See Figure 3 for an example of this. Note, for example, that the graph segment starting at 2b and going toward node three is not fully exposed.

The solutions to these problems may lie in finding a better way to represent the concealed regions in the graph.

7.2 Obstacle Avoidance

The current concealed-route planner assumes that all prospective paths are unobstructed. In actuality, it is very possible for a generated path to traverse obstacles such as rivers or steep slopes. Planning around obstacles and planning through concealed areas are currently done independently. The concealed route is modified to avoid obstacles after it has been planned. This can lead to sub-optimal routes because the concealed-route planner does not take the extra exposed distances imposed by obstacle avoidance into consideration. Ideally, one planner should be able to accommodate obstacles to avoid as well as attractive regions to traverse.

7.3 Considering Width

Currently, the concealed-route planner assumes that the vehicle or unit that travels along the route is infinitely thin. A vehicle located on the edge of a concealed region will actually be half-exposed. This problem can be solved by shrinking the concealed-region polygons by the half-width of the units following the routes.

8. Conclusions

One of the primary considerations in designing an algorithm in a CGF system is determining how well it fits in with the simulation. Specifically, does the algorithm cause long ticks? Overcoming the long-tick problem is especially challenging when computationally-expensive tasks, such as those which involve searching the terrain, need to be performed.

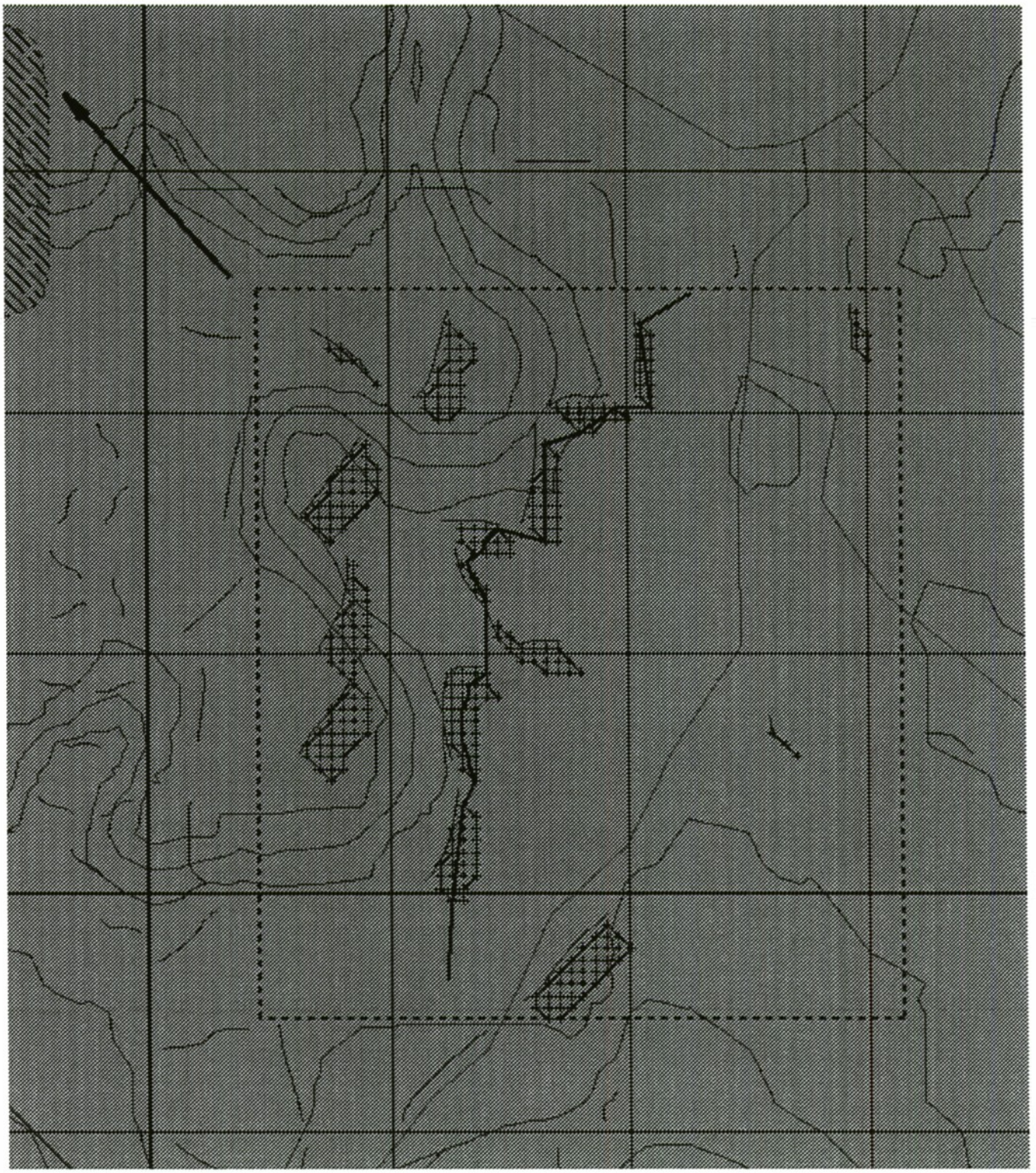


Figure 6: Concealed Routes Test Program

Concealment-map generation may take several seconds, depending on the size of the search area, and grid spacing, and the number of enemy descriptors. This problem is solved by using ModSAF's scheduler to distribute the search across multiple ticks, so that the rest of the simulation can run while the search is in progress. The route planner runs fast enough to be done in a single tick.

The concealed-route functionality has been tested both in the ModSAF system and by using a specially-designed test program, with promising results. A screen dump from the test program is shown in Figure 6. In this example, one enemy descriptor, an enemy direction, has been specified. This is denoted by the arrow. Each plus sign represents a "one" in the concealment-map grid, representing concealed regions. Note how the route (the black line) takes advantage of the available concealed regions.

The algorithms described in this paper are included in the terrain-reasoning library within the ModSAF system, and is comprised of roughly 2800 lines of C code.

The current version of ModSAF uses the concealed-route functionality in its bounding overwatch movement behavior and in rotary-wing aircraft nap-of-earth flight. Other behaviors currently implemented in ModSAF may be enhanced to take advantage of this functionality.

9. Acknowledgments

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058.

10. References

Barr A. and Feigenbaum, E. (1981). *The Handbook of Artificial Intelligence, Volume I*, Addison-Wesley.

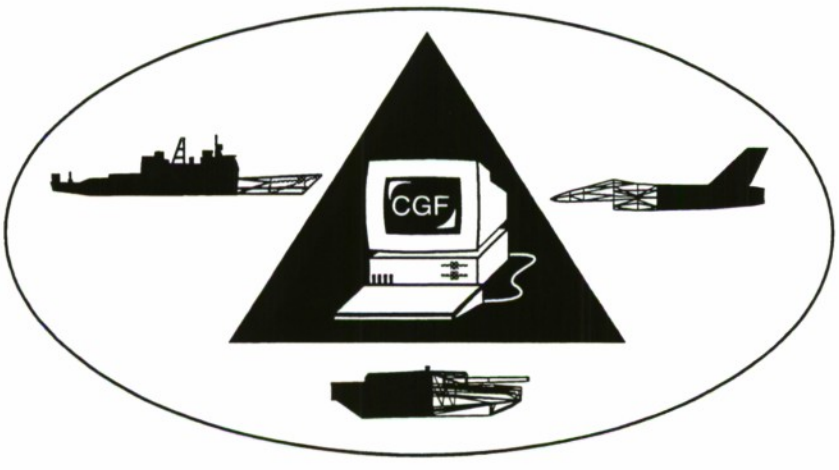
Longtin, M. J. (1994) *ModSAF Programmer's Guide: LibCTDB*, Loral Advanced Distributed Simulation, Cambridge, Massachusetts.

Tanimoto, S. L. (1987). *The Elements or Artificial Intelligence*, Computer Science Press.

11. Biographies

Michael J. Longtin graduated from the University of Maine with a B.S. in Electrical Engineering in 1991 and an M.S. in Electrical Engineering in 1993. His Master's thesis involved the design and development of an object-oriented robot programming language in C++. He has developed many programs using C in the areas of digital image processing, neural networks, and computer graphics. Since graduating, Michael has worked as a Software Engineer at Loral Advanced Distributed Simulation. He works at the Cambridge, Massachusetts office.

Dalila Megherbi graduated from Brown University with a Ph.D. in Electrical Engineering in 1993. Her research introduced to the robotics community an innovative approach for autonomous robot navigation based on the complex variable methodology. Her interests include computer-controlled robotics and automation, automatic control systems, and machine intelligence. Dalila joined Loral in September of 1994. She works at the Cambridge, Massachusetts office.


CGF

Terrain Avoidance for CGF Helicopters

Stephen A. Schricker, Robert W. Franceschini, Mikel D. Petty, Tracy R. Tolley
Institute for Simulation and Training
3280 Progress Drive, Orlando, Florida 32826-0544
sschrick@ist.ucf.edu

1. Abstract

The flight characteristics of a helicopter allow it to fly very close to the ground at relatively high speeds. Thus, the helicopter pilot, flying his or her aircraft at low altitude, must constantly adjust the helicopter's dynamics in order to prevent a disastrous and most likely lethal collision with the terrain. Simulating this process of terrain avoidance on a Computer Generated Forces system presents its own special challenges for the CGF programmer. This paper presents an algorithm for above-ground-level flight for helicopters; discusses its implementation in the IST CGF Testbed; discusses its use at the Aviation Testbed at Fort Rucker, emphasizing the interaction between CGF helicopters and U. S. Army soldiers flying manned helicopter simulators; and presents an analysis of the efficiency of the algorithm.

2. Background

Like many Computer Generated Forces systems, the focus in the development of the IST CGF Testbed was on ground combat (Smith 1992). The dynamics, combat, and behavioral models are quite detailed for tanks, infantry fighting vehicles, and dismounted infantry. Though the IST CGF Testbed initially contained some rudimentary algorithms for air vehicles, they were not as sophisticated as the algorithms for ground vehicles.

During 1994, the Integrated Eagle/BDS-D project (Franceschini 1995), which uses the IST CGF Testbed as a component, began using an armed reconnaissance scenario involving helicopters. A long-range goal was to use this scenario for training and analysis activities at Fort Rucker's Airnet site (a SIMNET facility).

IST installed a preliminary version of the Integrated Eagle/BDS-D system at Fort Rucker in July, 1994. During this installation, the Eagle project team observed the interactions between CGF-controlled helicopters and U.S. Army soldiers flying a manned helicopter simulator. It became clear that the CGF helicopters were not able to fly at low altitude as well as the U. S. Army pilots. The CGF helicopters frequently crashed when flying at low altitudes over mountainous terrain. This was due to the inadequacy of the simple above-ground-level flight algorithm in the IST CGF Testbed.

Late in 1994, the Eagle team presented the Integrated Eagle/BDS-D system in the Distributed Interactive Simulation (DIS) Interoperability Demonstration at the Interservice/Industry Training Systems and Education Conference (I/ITSEC). For the purposes of this demonstration, the CGF helicopters flew in an above-sea-level (ASL) mode which ignored the terrain under the vehicles and kept the helicopters at a constant height above sea-level. The heights of the helicopters were scripted prior to the demonstration rather than being calculated in real-time during the scenario. Although in this case the helicopters did not crash, the scripting of aircraft heights is clearly an undesirable strategem when dealing with supposedly autonomous entities.

To increase the realism of the virtual simulation, therefore, IST examined the problem of above-ground-level flight for CGF helicopters.

3. Above-Ground-Level Flight

A competent above-ground-level flight algorithm will allow helicopters to fly as fast and as close to the terrain as possible without crashing into the ground. One likely and easily implemented method (call it Algorithm S) would be to simply determine the height of the ground immediately underneath the helicopter and adjust the helicopter's altitude to maintain a predetermined above-ground-level height. Simple though Algorithm S may be, it is woefully inadequate. Since all helicopters have a limited rate at which they can climb–or descend, for that matter–the use of this method alone will produce disastrous results. There is no guarantee that the terrain will comply with the helicopter's physical limitations. For example, Figure 1 shows a helicopter moving forward at 30 m/s and climbing at its maximum rate of 12 m/s. Basic trigonometry defines the tangent of an angle to be equal to the magnitude of the y-component of the angle divided by the magnitude of the x-component of the

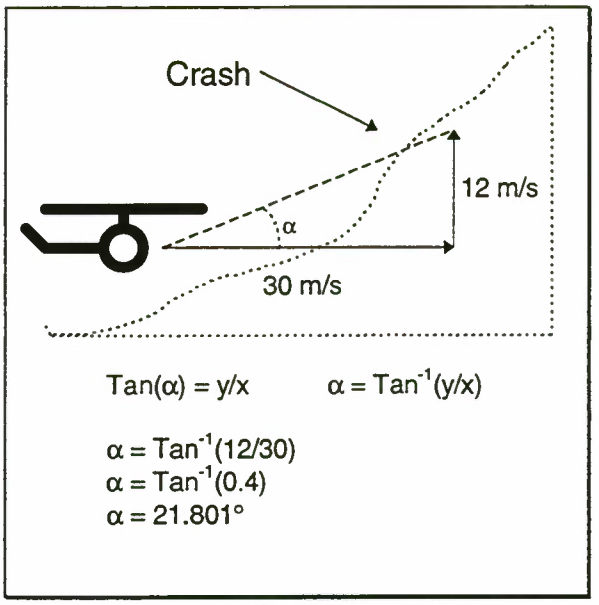


Figure 1

angle. Under these conditions, the helicopter will only be able to successfully traverse a grade of, at most, about 22 degrees. Any steeper grade will result in a collision with the terrain.

A variation on Algorithm S (say, Algorithm SL) might be to look ahead on the terrain some predetermined distance. This would give the helicopter a little extra time to account for steep terrain. Though the use of Algorithm SL could still yield results similar to Algorithm S, another not-so-evident shortcoming becomes apparent when the helicopter encounters a peak in the terrain.

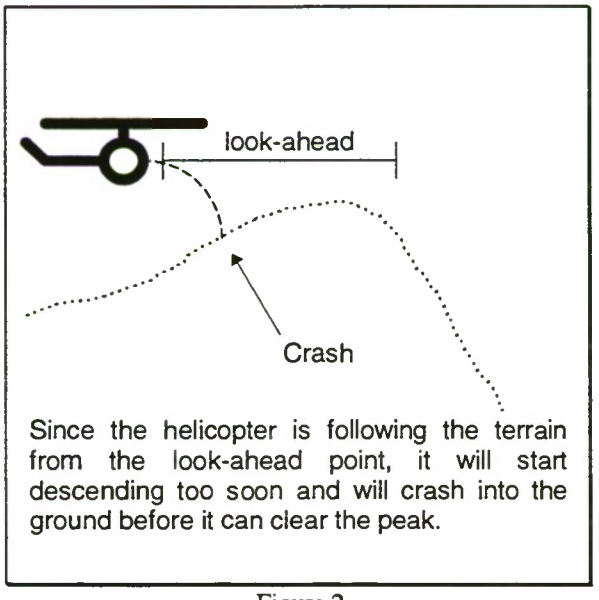


Figure 2

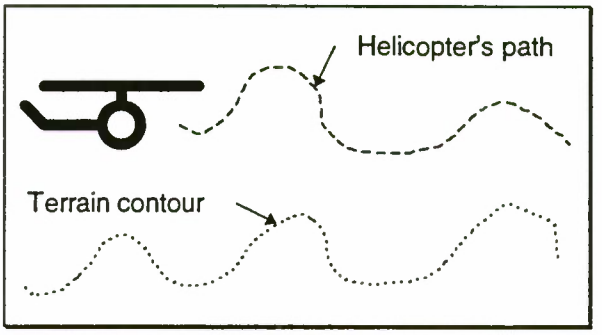


Figure 3

As Figure 2 shows, the slope on the far side of the peak could be steep enough to cause the helicopter to descend too early and too quickly, causing the helicopter to crash before it has cleared the peak.

Another major pitfall of both Algorithm S and Algorithm SL is that they tend to produce "bouncing" helicopters. By varying the helicopter's above-ground-level height with respect to a single point on the terrain, the helicopter's movement becomes sensitive to small variations in the contour of the terrain. In other words, with the helicopter adjusting its height according to every bump in the terrain along its path, as in Figure 3, it will "bounce" along as it flies.

4. A Mission Statement

To determine exactly what it is that an above-ground-level flight algorithm should and should not do, it is important to explicitly state the goals of such an endeavor. Of course, the ultimate goal is to produce realistic above-ground-level flight in helicopters; that is, to allow helicopters to fly as close to the terrain and as fast as possible, limited by the vehicle's dynamics and the algorithm's competence, while maintaining authentic flight characteristics. We can define the objectives that we need to satisfy in order to reach this final goal. First of all, the helicopters need to judge, in an empirical manner, the general trend of the terrain. Secondly, the helicopters need to determine whether they can adhere to the general trend of the terrain given their own design limitations. Finally, the method used to compel the helicopters into such behavior needs to be efficient and effective in order to maintain the integrity of the simulation.

5. IST's Above-Ground-Level Flight Algorithm

The phrase "general trend," repeated twice in the mission statement, suggests the application of statistical regression analysis to produce above-ground-level

flight in helicopters. Regression, in a statistical sense, concerns the prediction of the average value of one variable in terms of the known values of other variables (Freund 1979). In essence, regression is the process of "fitting" a curve to a set of points such that the curve describes, to a measurable degree of accuracy, the general trend of the set of points. In terms of above-ground-level flight for helicopters, regression analysis may be used to describe the general slope of the terrain along the helicopter's flight path.

5.1 Linear Regression Analysis

Linear regression is an analysis tool that attempts to fit a straight line to a set of sample points. Though there are more complex and accurate regression models that fit curved lines to a set of sample points, linear regression is computationally inexpensive, yet it is also sufficiently robust to accurately represent the small sample of terrain points that CGF helicopters need to establish above-ground-level flight. The time required to calculate the best-fitting line for the terrain sample is an order of magnitude proportional to n ($O(n)$), where n is the number of points in the sample. We will later show, however, that this calculation can be reduced to $O(1)$ on average. In addition, most terrain samples, regardless of the rate at which they slope, show a discernible linear trend when the bounds of the sample are relatively small, as is the case for IST's above-ground-level flight algorithm, presented below. This is especially true for a polygonal terrain database.

IST's above-ground-level flight algorithm for helicopters uses simple linear regression to determine the general slope of the terrain in front of the helicopter. It accomplishes this by sampling the upcoming terrain along the helicopter's current flight path. It then uses linear regression analysis to determine the general slope of the terrain given the set of terrain points that it has sampled. By determining this general slope, the algorithm can adjust the helicopter's dynamics values to reflect changes in the terrain in front of the helicopter. In essence, regression analysis produces a generalized "picture" of the terrain in front of the helicopter, giving equal weight to every point sampled along the terrain, thus diminishing the effect of minor terrain contour variations on the helicopter's flight. This produces smooth flight characteristics while allowing the helicopter to simultaneously conform to the terrain and account for its own physical limitations.

The algorithm utilizes the method of least squares to determine the best-fitting straight line in which to model the terrain. The method of least squares computes a straight line such that the sum of the squares of the vertical deviations of the sample points from that line is a minimum (Freund 1979).

5.2 The Least-Squares-Determined Line

The equation for a straight line represented in the Cartesian coordinate system, shown in the top part of Figure 4, takes the form $y=a + bx$, where a is the y-intercept and b is the slope of the line. Given two points on the line, the line's slope is determined by dividing the difference in the y-values of the two points by the difference in the x-values of the two points. Now, given just one (x, y) coordinate pair and the slope of the line, it is possible to compute the y-intercept of the line.

Equation for a line: $y = a + bx$ (1)
a equals the value for y when x=0.
b equals the magnitude of the slope.

Equation for the slope of the line from (x_1, y_1) to (x_2, y_2):

$$b = (y_2 - y_1) / (x_2 - x_1)$$

Equation for the y-intercept of a line (from Equation (1) above):

$$a = y - bx$$

Equation for the average slope of a curve using the least-squares method:

$$b = \frac{n(\sum xy) - (\sum x)(\sum y)}{n\left(\sum x^2\right) - (\sum x)^2}$$

Equation for the y-intercept of a curve using the least-squares method:

$$a = \frac{(\sum y)\left(\sum x^2\right) - (\sum x)(\sum xy)}{n\left(\sum x^2\right) - (\sum x)^2}$$

n is the number of sample points
Σx is the sum of all x values.
Σy is the sum of all y values.
Σxy is the sum of all x values multiplied by their respective y values.
Σx^2 is the sum of the squares of all x values.

Figure 4

Computing the slope and y-intercept values for a least squares line is not so straightforward, however, as the

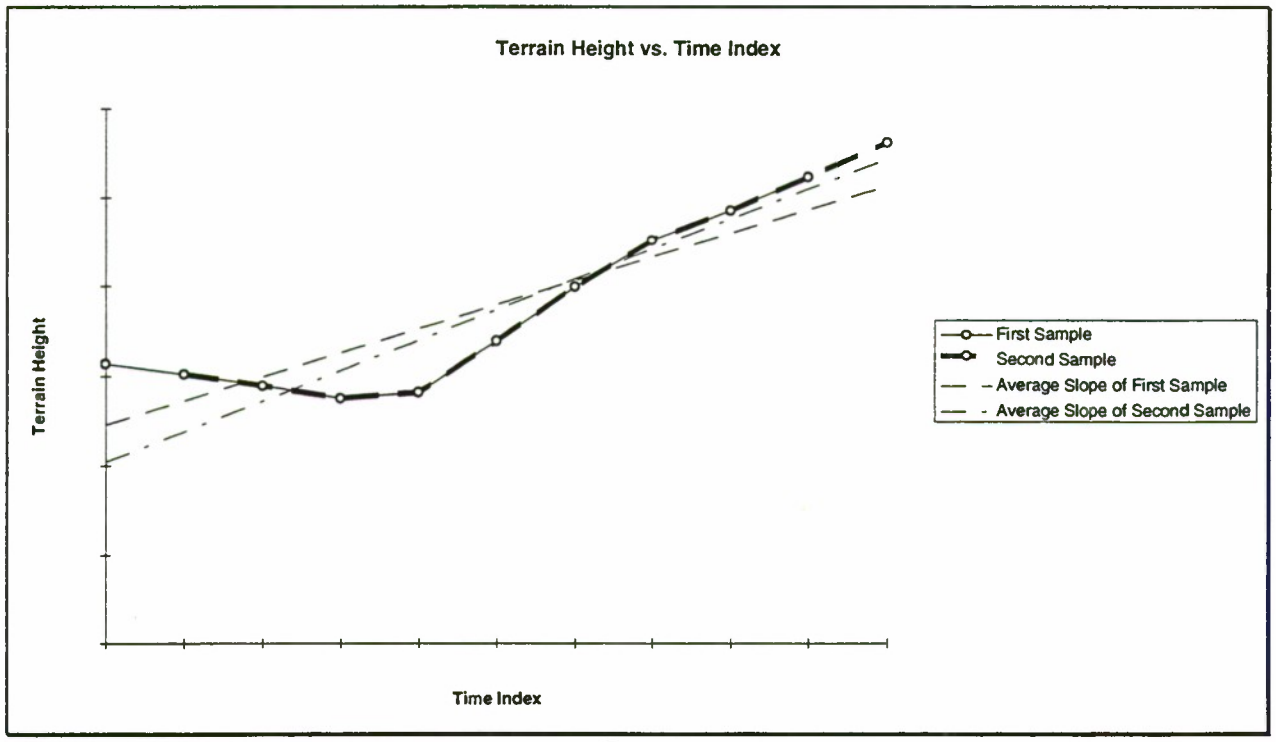


Figure 5

```
PerformAnalysis();       /* Perform regression analysis. */
predicted_height = InterpolateGroundHeight;

/* Perform altitude adjustment. */
if (predicted_height > sample_max)
    /* Change height relative to predicted height. */
else
    /* Change height relative to sample_max.       */

/* Perform speed adjustment. */
grade_factor = 1 - (current_speed / max_speed);

if (terrain_slope != 0.0 &&
    (terrain_slope > (grade_factor * MAX_HEIGHT_CHANGE_RATE)))
{
    speed_adj_ratio = (grade_factor * MAX_HEIGHT_CHANGE_RATE) /
                      terrain_slope;
    current_speed = current_speed * speed_adj_ratio;
}
else if (heli_altitude > sample_max)
    current_speed = initial_requested_speed;
```

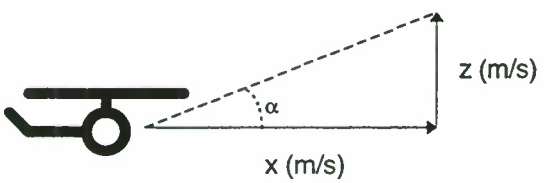
Figure 6

bottom part of Figure 4 shows. For a more complete discussion on the determination of the least squares line, see (Freund 1979).

Figure 5 shows the least squares line determined for two successive samplings of the terrain. Notice how the magnitude of the slope of the line changes gradually from one sample to the next. The application of this principle not only produces smoother, more realistic flight, but it also allows the helicopter to make other adjustments, based on the magnitude of the slope, to account for the helicopter's design limitations.

5.3 How Does the Algorithm Work?

The above-ground-level flight algorithm, shown in Figure 6, performs its terrain analysis every time the helicopter's dynamics are updated. `PerformAnalysis()` performs the actual linear regression analysis, computing the intercept (*a*) and the slope (*b*) of the least-squares line for the current terrain sample. Given the values for *a* and *b*, `InterpolateGroundHeight()` predicts the height of the terrain along the helicopter's path at a pre-determined point in time. The algorithm then performs simultaneous adjustments to both the helicopter's altitude and its speed. The altitude adjustment requires little explanation since the goal of this algorithm is to compel the helicopter *over* the terrain–thus, the need for the altitude adjustment. The speed adjustment, however, is what takes into account the physical limitations of the helicopter and its ability to change its altitude.



Given:

1) $x \geq 0; z \geq 0$
2) $0° \leq \alpha < 90°$

$\tan(\alpha) = z/x$

Suppose that x varies while z is constant. As $x \to 0$, $z/x \to +\infty$ (since $z>0$ and $x>0$). Thus, $\tan(\alpha) \to +\infty$.

Since $0° \leq \alpha < 90°$, we know that, as $\tan(\alpha) \to +\infty$, $\alpha \to 90°$, because on the interval $0° \leq \alpha < 90°$, α increases as $\tan(\alpha)$ increases.

Therefore, the maximum climbing angle increases as the forward speed of the helicopter (and hence, x) decreases.

Figure 7

It is important to note here the relation between forward speed and climbing rate. Using trigonometry, it is possible to determine the maximum slope that a helicopter can traverse given its forward speed and its maximum climbing rate. In addition, as Figure 7 shows, when the helicopter's speed decreases, the maximum slope that the helicopter can traverse increases. In Figure 7, x represents the helicopter's forward speed, z represents the helicopter's climbing rate, and α represents the angle formed by those two components of the helicopter's motion. In this example, the following conditions exist: the helicopter is moving forward at some positive rate ($x \geq 0$), it is climbing at some positive rate ($z \geq 0$), and therefore, the angle formed by the two components of the helicopter's motion (α) is between 0° and 90°. The tangent of α is equal to z/x. Now, suppose that z remains constant while x changes (which corresponds to decreasing the helicopter's forward speed while allowing its climbing rate to remain constant). As x decreases, α increases and finally approaches 90°. Therefore, the helicopter's maximum climbing angle increases as its forward speed (and hence, x) decreases, which means that slowing down the helicopter makes it better able to avoid steep terrain.

The helicopter performs its terrain analysis based on its sample of terrain heights at various points along the helicopter's flight path, and the time at which the helicopter recorded each sample point. Using the least squares method for linear regression analysis, the helicopter determines the average slope, in meters per second, of the terrain in front of it. This average slope shows how quickly the helicopter must climb in order to remain at a relatively uniform height above the terrain. Based on the computed average slope of the terrain, the helicopter predicts a height for the terrain at a pre-determined point along its flight path. The helicopter adjusts its altitude to reflect a given desired height above the ground relative to the height of the terrain.

5.4 Why Adjust the Helicopter's Speed, as Well?

There are two valid reasons behind the speed adjustment. First, we have already shown that a speed adjustment is necessary to account for the physical limitations of the helicopter. Secondly, although changes in the helicopter's dynamics settings as a result of the algorithm do occur instantaneously, the behaviors that reflect such changes are not instantaneous. In addition to accommodating the physical limitations of the helicopter, the speed adjustment provides a buffer to take into account the time that it takes for the helicopter to reflect in its behavior any changes in its dynamics.



Suppose:

MAX_SPEED = 120 m/s
MAX_DELTA_Z = 12.0 m/s

$$speed_ratio = \frac{CURRENT_SPEED}{MAX_SPEED}$$

$$speed_ratio = \frac{75\ m/s}{120\ m/s}$$

speed_ratio = 0.625

grade_factor
= 1 - speed_ratio
= 1 - 0.625
= 0.325

Slope threshold
= 0.325 * MAX_DELTA_Z
= 0.325 * 12.0 m/s
= 3.9 m/s

Therefore, in this example, if the terrain slope is greater than 3.9 m/s, the algorithm will decrease the speed of the helicopter proportional to the ratio of the slope threshold to the terrain slope.

Figure 8

From the pseudo-code in Figure 6, the altitude adjustment is rather simple. The helicopter merely adjusts its altitude to reflect changes in the height of the terrain. The speed adjustment is a little more complex, however. From Figure 8, the helicopter first computes its speed ratio–the ratio of its current speed to its maximum attainable speed. The helicopter uses the speed ratio to compute the grade factor. The grade factor gives the helicopter the buffer that it needs to account for the time that it takes to reflect in its behavior any changes in its dynamics. Thus, the grade factor is inversely proportional to the speed of the helicopter relative to its maximum speed (computed by subtracting the helicopter's speed ratio from 1). The slope threshold is then computed from the grade factor and the maximum rate at which the helicopter can climb. Therefore, the point at which the helicopter begins adjusting its speed is inversely proportional to the relative speed of the helicopter. In other words, the faster the helicopter is moving, the earlier it will begin adjusting its speed. This process of sampling and dynamics adjustment is continuous.

6. Implementation in the IST CGF Testbed

The IST CGF Testbed is designed such that, under all but the very worst conditions, the dynamics model of any entity is updated roughly five times per second. This greatly simplifies the sample-gathering process. Rather than gathering an entire sample during each iteration of the dynamics update, the helicopter merely disposes of the oldest sample point and all traces of it and replaces it with a new one. However, before destroying the oldest sample point, its components are subtracted from Σx, Σy, Σxy, and Σx^2, and the values for the new sample point are then added to what remains. The helicopter must gather an entirely new set of sample points only when it changes its angular velocity. Thus, the gathering of the sampling data and the performance of the regression analysis is *O(1)*, on average.

IST determined that a five-second look ahead would be sufficient to accurately model the terrain for the purpose of establishing above-ground-level flight. This determination was made based on an interpretation of when a real helicopter pilot would begin adjusting his or her aircraft to avoid the terrain. Given this five-second look-ahead, and the fact that the helicopter's dynamics model is updated about five times per second, the above-ground-level algorithm uses a sample size of twenty-five points with which it performs its regression analysis. The projection of the helicopter's position five seconds into the future is a simple calculation given the helicopter's velocity and its current speed.

In order to keep track of all of the data required to establish above-ground-level flight, it was necessary to add some new data structures, shown in Figure 9, to the entity control structure for helicopters.

`GROUND_HEIGHT_DATA` is the structure which holds the data for the computation of the least squares line for the terrain sample, and consists of the following:

> `sample` is an array which holds the ground-height/time-index sampling pairs.
>
> `head` points to the oldest sample in the array. The sample data is implemented as a circular list, which makes it easy to determine which sample point to discard when collecting a new sample point.

```
typedef struct      /* Helicopter Ground Sample */
{
    double ground_height;       /* Ground height of sample */
    double time_index;          /* Time index of sample    */
} GROUND_SAMPLE;

typedef struct      /* Helicopter Ground Sampling Data */
{
    GROUND_SAMPLE
        sample[TERRAIN_SAMPLE_SIZE];    /* Terrain Sample                 */
    int head;                           /* Head of sample list            */
    double gh_mean_slope;               /* Least-squares determined slope */
    double gh_intercept;                /* Least-squares determined int.  */
    double gh_max;                      /* Max. height value in sample    */
    double sigma_x;                     /* Sum of x (time_index)          */
    double sigma_y;                     /* Sum of y (ground_height)       */
    double sigma_xy;                    /* Sum of x*y                     */
    double sigma_x_squared;             /* Sum of x^2                     */
    TIME latest;                        /* Most recent time index         */
} GROUND_HEIGHT_DATA;
```

Figure 9

`gh_mean_slope` and `gh_intercept` describe the least squares line. The equation for the line is of the form `y=a + bx`, where `a` is the y-intercept, which corresponds to `gh_intercept`, and `b` is the slope of the line, which corresponds to `gh_mean_slope`.

`gh_max` stores the maximum terrain height value in the sample. By comparing the predicted height of the terrain with the maximum height in the sample, the helicopter has a buffer to prevent collisions with the tops of terrain peaks.

`sigma_x`, `sigma_y`, `sigma_xy`, and `sigma_x_squared` are used in the computation of `gh_mean_slope` and `gh_intercept`. These variables correspond to Σx, Σy, Σxy, and Σx^2, respectively, in the regression analysis. They are stored in the helicopter's control structure to optimize the computation of the least squares line.

`latest` records the time at which the latest sample point was recorded.

6.1 Minor Algorithm Modifications

In the real world, flying a helicopter–or any aircraft, for that matter–just above the treetops and at maximum speed is a risky venture, to put it mildly. The probability of the flight ending in a fatal collision with the ground is very real. The cost of such a crash in the virtual world may not be as high, but the goal of the simulation is to make the consequences just as real. Therefore, any CGF helicopter that crashes into the ground due to the inadequacies of its above-ground-level flight algorithm is unacceptable. The algorithm must be perfect in the sense that it does not allow *any* helicopter to crash as a result of the characteristics of the terrain.

It is for this reason that the above-ground-level flight algorithm presented here implements certain measures that cause the helicopters to behave more conservatively with respect to their flight characteristics. Regression analysis is able to base its conclusions only on the sample on which it has to operate. With these conclusions comes a measurable degree of accuracy, which means that they are not always the correct conclusions. Any analysis performed on the terrain with the intent to produce above-ground-level flight in helicopters must always arrive at the correct conclusion, however. Thus, in addition to the terrain height sample, the helicopter also stores the maximum value of the terrain height in the sample. In adjusting its altitude, the helicopter proceeds to its given height above the ground plus the greater of either the predicted terrain height or the maximum terrain height in the sample.

The use of the grade factor in the adjustment of the helicopter's speed performs an analogous function.

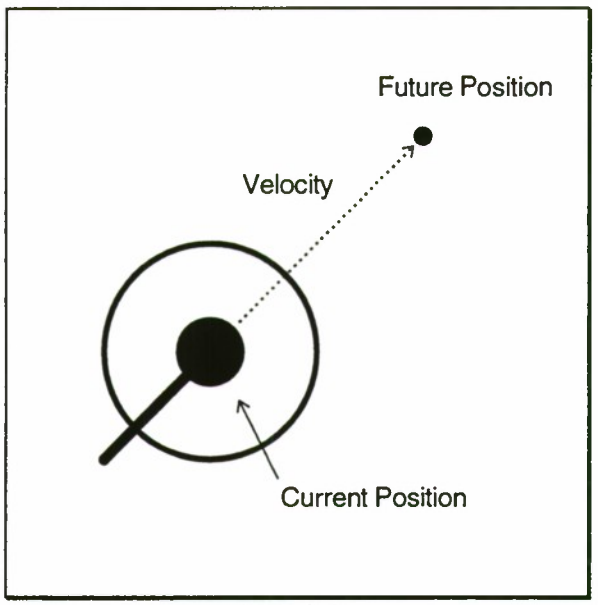


Figure 10

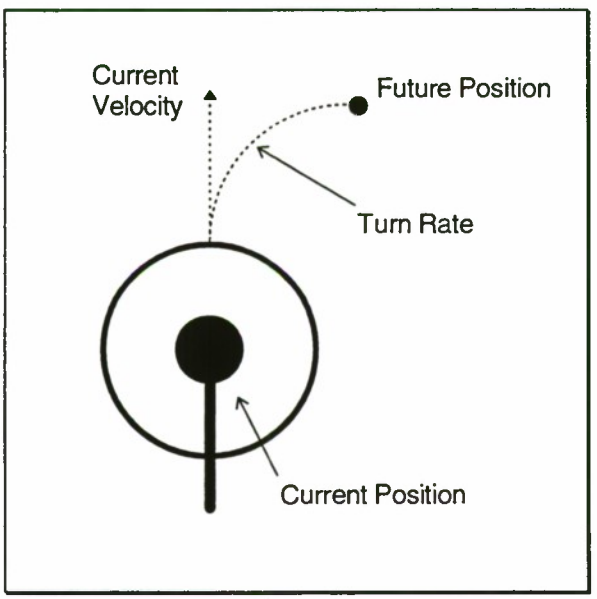


Figure 11

6.2 Curved Flight-Path Projection

When the helicopter is flying in a straight line, as in Figure 10, the projection of its path along that line is a simple calculation given the helicopter's current position, its velocity (remember from physics, a body's velocity consists of its speed *and* direction), and the time displacement of the projection–the length, in time, of the look-ahead. It would be unrealistic and would most likely yield unacceptable results to project a straight-line position when the helicopter is actually following a curved, or more precisely, circular path, as in Figure 11. It becomes necessary, therefore, to

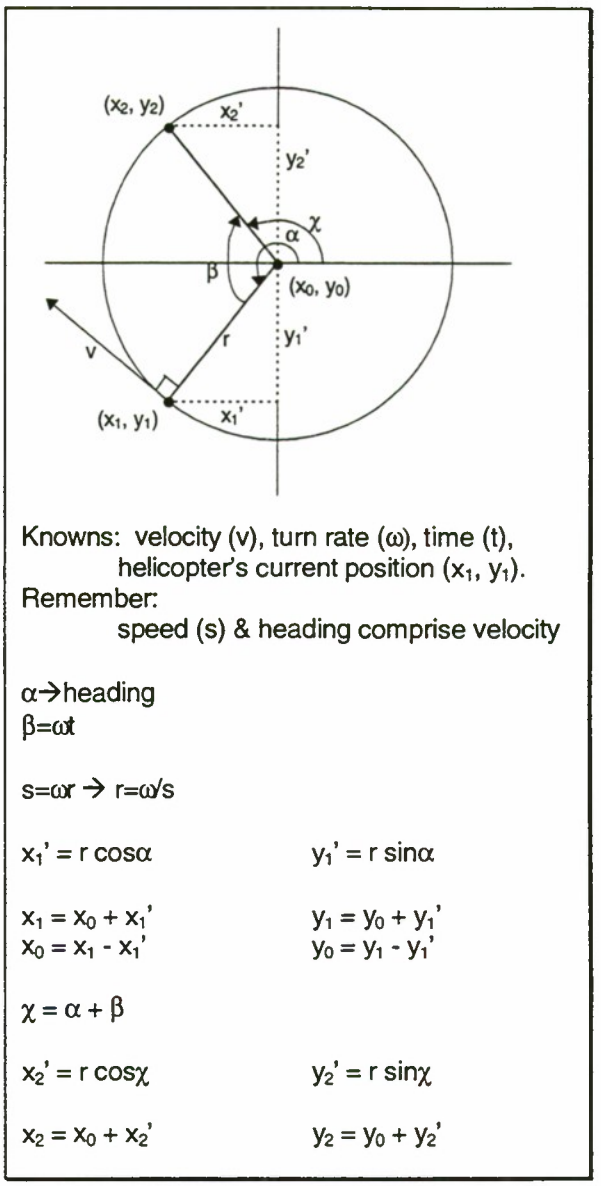


Figure 12

project the helicopter's position along a circular path, given a constant angular velocity, in order for it to gather a valid terrain sample along that path.

In projecting the helicopter's position along a straight-line path, it is necessary to know the helicopter's current position, its velocity, and the length of the look-ahead time. In projecting the helicopter's position along a circular path, it is also necessary to know the rate at which the helicopter is turning. Given these known values, Figure 12 shows the method by which the helicopter projects its position along a circular path.

From Figure 12, the helicopter knows its current position (x_1, y_1), its speed (s) and its heading (determined directly from its velocity [v]), its turn rate

(ω), and the time displacement for the projection (t). The projection of the helicopter's position along its circular path through t requires two steps. First, the helicopter must determine its center of motion (x_0, y_0). Given (x_0, y_0), it can then proceed to project its future position (x_2, y_2).

To compute (x_0, y_0), the helicopter must first "normalize" its heading relative to (x_0, y_0). (x_0, y_0) will be different depending on whether the helicopter is moving in a clockwise fashion or a counter-clockwise fashion, determined by the sign of ω. A positive ω depicts clockwise motion. Figure 12 shows the helicopter moving in a clockwise fashion. The normalized heading (α) is determined directly from the helicopter's actual heading and the value of ω.

Given ω and s, the helicopter can compute the radius (r) of the circle defined by its motion. Now, after computing α and r, the helicopter can determine (x_0, y_0) given (x_1, y_1).

To perform the actual projection of the helicopter's position along its current circular flight path, the helicopter first calculates, directly from ω and t, the magnitude of the angle (β) through which it passes during t. χ, the normalized value for β, is determined simply by adding α and β. Finally, after computing χ, the helicopter can now determine its future position, (x_2, y_2), given (x_1, y_1).

7. Fort Rucker's Aviation Testbed

The original scenario installed at Fort Rucker in July, 1994, was criticized for its lack of tactical realism. The main reason for this was that the CGF helicopters were unable to fly close to the ground without crashing. In order to make it through the scenario without colliding with the terrain, however, the helicopters needed to be placed at very high altitudes–an unrealistic and tactically unsound enterprise.

In early 1995, IST installed an updated version of the Integrated Eagle/BDS-D system at Fort Rucker. Among the revisions was a new training and analysis scenario and an improved CGF Testbed which included the above-ground-level flight algorithm for helicopters.

In the revamped scenario, the above-ground-level algorithm allows the use of CGF helicopters in a more tactically feasible manner.

8. CGF/Manned Simulator Interaction

The Integrated Eagle/BDS-D system has the capability to instantiate manned simulators in the virtual environment, allowing the interaction between CGF entities and U.S. Army soldiers operating manned simulators. In the original installation of the Integrated Eagle/BDS-D system at Fort Rucker, the scenario normally proceeded with a manned helicopter simulator flying extremely low to the ground at a relatively high speed while the CGF helicopters in the scenario followed at very high altitude. The soldiers operating the manned helicopter simulator, and observers alike, readily noticed this unrealistic formation and criticized the CGF helicopters' behavior. With the new above-ground-level flight algorithm, however, the CGF helicopters were able to fly as low as the pilots without crashing, though not quite as fast.

9. Algorithm Efficiency

This algorithm has a distinct advantage over similar algorithms due to its size and efficiency. By storing the Σ-values as part of the helicopter's control structure, complete recomputation of these values is not necessary. Only when the helicopter changes its angular velocity and, therefore, gathers a new sample, does it need to perform an entire recomputation of the Σ-values. Therefore, on average, the regression analysis has an order of magnitude of one ($O(1)$).

The determination of the maximum terrain height value in the sample is also $O(1)$ on average. Recall that when a new sample point (p_{new}) is gathered, the oldest point in the sample (p_{old}) is removed from the sample. To determine the maximum height in the sample, p_{new}'s terrain height is compared with the stored maximum height of the sample (if p_{new}'s height is larger than the maximum, then p_{new}'s height becomes the new maximum). This operation requires $O(1)$ steps. However, if p_{old}'s terrain height is the maximum height of the sample, then the algorithm must recompute the maximum height of the sample by examining each point in the sample. Although this operation takes $O(n)$ steps, it is performed infrequently as compared to the $O(1)$ calculation above; therefore, the height determination for the sample takes $O(1)$ steps on average.

10. Limitations and Future Work

The above-ground-level flight algorithm presented here does have its weaknesses, however. The *Assault Support Helicopter Tactical Manual* (Department of the Navy 1984) describes three types of flight that a

helicopter might be required to perform: low-level, contour, and nap-of-the-earth. Low-level flight places the helicopter just above the terrain as a whole, allowing it to fly at its absolute maximum speed without regard to the terrain. Contour flight places the helicopter much closer to the terrain, making it necessary for the helicopter to take into account the terrain in order to avoid crashing. The above-ground-level flight algorithm presented here most closely mimics contour flight. Nap-of-the-earth flight involves flying "as close to the Earth's surface as vegetation and obstacles permit while generally following the contours of the Earth's surface" (Department of the Navy 1984). Nap-of-the-earth flight techniques are usually limited to missions in which the threat and tactical situation preclude the safe use of low level or contour flight techniques. This technique greatly sacrifices speed, but it takes advantage of the cover and concealment afforded by terrain, vegetation, and manmade features.

The Integrated Eagle/BDS-D system utilizes contour flight for its CGF helicopters simply because the system's demonstration scenario calls for that particular flight mode. To produce a more realistic simulation environment, however, the system should implement both low-level and nap-of-the-earth flight, as well. The human operator could specify the desired flight mode of a CGF helicopter upon its instantiation. Each mode might define a minimum height at which it would allow helicopters to fly. In addition, the operator could switch flight modes for specific helicopters during the simulation. Ideally, however, a CGF helicopter would vary its flight mode automatically and intelligently, depending on its environment or on orders from higher headquarters.

10.1 Operational Activities in Eagle/BDS-D

The Integrated Eagle/BDS-D Project has successfully linked a constructive, aggregate-level simulation with a virtual, entity-level simulation using both manned simulators and computer generated forces (Franceschini 1995). When units from the constructive simulation are disaggregated as computer generated forces into their respective vehicles in the virtual simulation, they are given operational activities upon which they are to act in an intelligent manner. Among the operational activities for helicopters are "FLY-NOE" and "FLY-CONTOUR", telling the CGF helicopters the mode in which they will be flying. In such a system, those helicopters should be placed at altitudes corresponding to their operational activities: perhaps 100 to 150 meters for low-level flight; 20 to 30 meters for contour flight; 5 to 10 meters for nap-of-the-earth flight.

10.2 Intelligent Route Planning

Thus far, the subject of this paper has focused solely on the adjustment of the helicopter's height (and its speed; but only for the purposes of allowing the helicopter to climb at a steeper rate). A more autonomous CGF helicopter would adjust its route as well in response to its environment. For example, it is more tactically sound for a helicopter to fly through a canyon than to fly outside of one. Therefore, a CGF helicopter should analyze the terrain with regards to the mode in which it is flying to take full tactical advantage of the surrounding environment. Such analysis is beyond the scope of the above-ground-level flight algorithm presented here, and would require an intelligent route planning algorithm not unlike that which is in place for ground vehicles in the IST CGF Testbed (Smith 1992). Route planning occurs before the helicopter even starts moving, while the above-ground-level flight algorithm presented here adjusts the helicopter en route. Hence, the two algorithms would be able to work independently, but for the same higher goal: realistic and tactically sound helicopter flight.

At this time, however, the scope of the Integrated Eagle/BDS-D Project does not encompass these types of intelligent flight controls for CGF helicopters.

10.3 Future Work

Future work in the area of above-ground-level flight for CGF helicopters should include the implementation of low-level and nap-of-the-earth flight modes; intelligent interpretation of operational activities, especially for constructive+virtual linkages, and intelligent route-planning that accompanies both contour and nap-of-the-earth flight.

11. Acknowledgment

This research was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command and by the U.S. Army TRADOC Analysis Center as part of the Integrated Eagle/BDS-D Project, contract number N61339-92-K-0002. That support is gratefully acknowledged.

Special acknowledgment goes to Robert C. Schricker, who provided invaluable assistance in the determination of the curved-flight-projection routine.

12. References

Department of the Navy (1984). *Assault Support Helicopter Tactical Manual NWP 55-9-ASH*,

Volume I (Rev. B) NAVAIR 01-1ASH-1T, Office of the Chief of Naval Operations.

Franceschini, Robert W. (1995). "Integrated Eagle/BDS-D: A Status Report", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, May 9-11, 1995.

Freund, John E. (1979). *Modern Elementary Statistics*, Fifth Edition, Prentice-Hall.

Smith, Scott H., Karr, Clark R., Petty, Mikel D., Franceschini, Robert W., Wood, Douglas D., Watkins, Jon E., and Campbell, Charles E. (1992). "The IST Semi-Automated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, University of Central Florida.

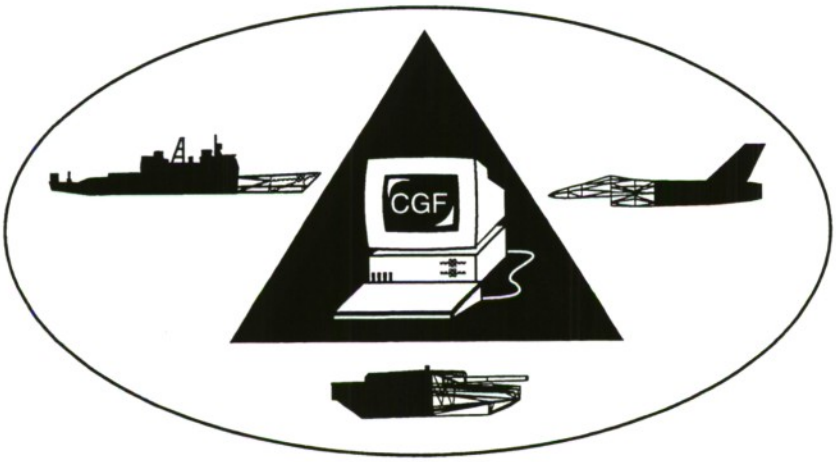
13. Biographies

Stephen A. Schricker is a Software Engineer on the Integrated Eagle/BDS-D Project at the Institute for Simulation and Training. He has earned a B. S. in Computer Science from the University of Central Florida. His research interests are in the area of simulation.

Robert W. Franceschini is a Principal Investigator at the Institute for Simulation and Training. He currently leads the Integrated Eagle/BDS-D project at IST. Mr. Franceschini has earned a B. S. in Computer Science from the University of Central Florida; he is currently pursuing an M. S. in Computer Science from UCF. His research interests are in the areas of simulation, graph theory, and computational geometry.

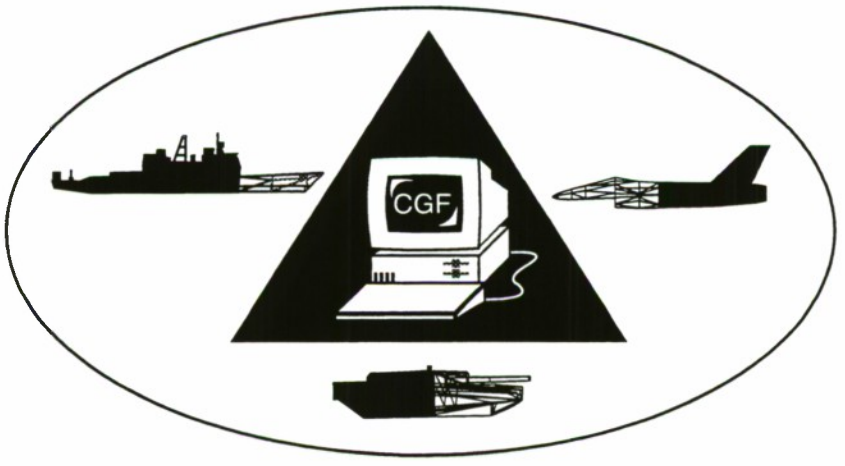
Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He is currently managing Plowshares, an emergency-management simulation project. Previously, he led IST's Computer Generated Forces research projects. Mr. Petty has earned a B. S. in Computer Science from California State University at Sacramento, and an M. S. in Computer Science from the University of Central Florida. He is currently a Ph. D. student in Computer Science at UCF. His research interests are in the areas of simulation and artificial intelligence.

Tracy R. Tolley is a Graduate Research Assistant on the Integrated Eagle/BDS-D project at the Institute for Simulation and Training. She has earned a B. S. in Mathematics from the University of Central Florida, and is currently pursuing an M. S. in Computer Science at UCF. Her research interests are in the area of simulation.


CGF

Session 7a: Non-Military Uses of CGF

Warren, GreyStone Technology, Inc.
Petty, UCF/IST
Moore, University of Pennsylvania


CGF

Bi-Directional Technology Transfer Between Government Applications of Computer Generated Agents and Commercial Entertainment

Rich Warren
Mike Crowe
Don Shillcutt
GreyStone Technology, Inc.
10766 Goldentop Road
San Diego CA 92127
rwarren@gstone.com

1. Abstract

The transfer of Computer Generated Forces (CGF) technology between government simulation and commercial entertainment communities, facilitates the development of more evolved and cost effective Autonomous Intelligent Adversaries (AIA). As commercial AIA requirements begin to also meet government CGF requirements, breakthroughs in intelligent adversary technology are incorporated into Commercial Off The Shelf (COTS) and Value Added software. The commercial *reusable* software tools are in turn made available to government CGF managers who realize immediate reductions in development costs and program maintainability.

This paper will describe early application of CGF technology to both government simulation and commercial gaming environments. More recent applications of the technology will also be discussed to show that the fidelity requirements of AIA in simulation and gaming are, by today's standards, nearly identical. Motivation to *reduce* the development, acquisition and operations cost of CGF and AIA software tools that *increase* the fidelity, performance and portability of behavior models is also offered.

2. Common Vision

The Distributed Interactive Simulation (DIS) glossary defines CGF and Semi-automated Forces (SAFOR) as the "Simulation of friendly, enemy and neutral entities on the virtual battlefield in which the individual platforms are operated by computer simulation of the crew and command hierarchy." The Virtual or *Electronic* Battlefield is likewise defined as the "Illusion resulting from simulating the actual battlefield (IST, 1994)."

The application of CGF technology in government military Test, Training and Analysis exercises satisfies the government's need to reduce cost and logistics support while maintaining the density, depth and diversity of forces required to accomplish the exercise objectives. Though the human or *live* forces in the exercise remain the focal point, the use of CGF provides an economic solution that stimulates interactions between "players" on the virtual battlefield.

The commercial entertainment industry, like the military, has similar needs for an economic solution that stimulates live (i.e., cash paying) customers. The profit for commercial entertainment is derived from enticing the customer to participate in and repeatedly return to the gaming environments and location based experiences . Commercial Virtual Reality opportunities are growing through the application of technology that offers a solution.

The DIS glossary defines VR as the "effect created by generating an environment that does not exist in the real world. Usually, a stereoscopic display and computer-generated three-dimensional environment giving the immersion effect. The environment is interactive, allowing the participant to look and navigate about the environment, enhancing the immersion effect. Virtual environment and virtual world are synonyms for virtual reality (IST, 1994)."

Notice that a "virtual battlefield" is one representation or application of "virtual reality." GreyStone Technology's commercial virtual reality entertainment systems combine multi-sensory human-computer interfaces with real-time simulations and dynamic models that display intelligent and interactive behaviors

(Crowe, M., 1994). GreyStone's VR entertainment systems represent commercial CGF applications that are strikingly similar those required by the military.

As outlined above, government simulation and commercial entertainment managers now share a common goal to reduce the cost of development, acquisition and operation of CGF technology. In the sections that follow, early applications of CGF technology to both government simulation and commercial gaming environments will be presented to show that CGF requirements across the two environments were at one time distinct. More recent applications of CGF technology, however, will also be discussed to show that the fidelity requirements of Intelligent Adversaries in simulation and gaming are by today's standards nearly identical. Closing sections of this paper will offer motivation to *reduce* the development, acquisition and operations cost of CGF and AIA software tools that *increase* the fidelity, performance and portability of behavior models.

3. Early Applications of CGF Technology

Until recently, user requirements across government simulation and commercial gaming environments have placed differing emphasis on the fidelity of the CGF and AIA. While the government simulation environments required high fidelity CGF , the commercial gaming environments required high fidelity presentation systems. In the sections that follow, several exemplar applications of early work from both government simulation and commercial gaming will be presented to show that computational resources were not sufficient to simultaneously host both CGF and display technology.

3.1 Early Government Simulation

The government has shown an interest in modeling adversarial forces since WWII. Much of modern CGF technology can in fact be traced back to the work of John Von Neumann and his theories of game play (Von Neumann, 1944). This section will trace early work in game theory and Operations Research to motivate the discussion of applications that show government's early emphasis on CGF.

Adversarial Agent Modeling and *Computer Generated Vehicle Commanders* are applications that are described to show an early government emphasis on CGF technology rather than on presentation technology. Following, GreyStone Technology's *Advanced Maneuvering Logic - 90* (AML-90) (GreyStone, 1994) will be presented to illustrate a government interest in CGF technology hosted in a computational environment sufficient enough to also provide two dimensional bitmapped graphics.

3.1.1 Operations Research

Operations Research is an activity with a long history that dates back to World War II. Methods of Operations Research, including statistical analysis, theory of probability and gaming theory, have been applied to tactical analysis and operational experiments with equipment and procedure for over half a century. (Morse and Kimball, 1951).

Tactical analysis became necessary as the onset of WWII introduced many new tactics and equipment types for which effective measure-counter measures were needed. A counter measure to minimize the threat of the Japanese Kamikaze, for example, was found through statistical solutions that considered damage due to suicide planes, the effects of maneuvering and the effect of *angle of approach.* The results of the Kamikaze study produced a number of suggested tactics which resemble, in content, the *consequent* of modern day expert system rules. Below is a summary table of some of the tactics learned through the statistical analysis of suicide planes.

Rule No	Tactic Learned
1	All ships should attempt to present their beams to high-diving planes.
2	All ships should attempt to turn their beams away from low-diving planes.
3	Battleships, cruisers, and carriers should employ radical changes of course
4	Destroyers and smaller fleet units should turn slowly to present the proper aspect to the diving plane.
5	Destroyers and smaller fleet units should not turn rapidly enough to affect the accuracy of their AA.

Table 1: Tactics Learned

In addition to statistical analysis, *search* and *game* theories were developed to provide more analytical solutions to tactical analysis. Search theories, for example, can state the probability of making contact with a target placed at random within some given area. The *probability*

of hit (Pk) is likewise computed using statistical theory.

Game theories were also developed as problem solving methodologies to tactical analysis. Specifically, the analysis of countermeasure action is accomplished using principals established by Von Neumann. These principals are particularly effective under situations where battlefield intelligence is complete and the opposing forces are reasonably familiar with measures and countermeasures that apply to the tactical situation. The driving principal under such situations, know as the *minmax principal*, works to maximize gain while minimizing loss.

3.1.2 Enemy Platforms

Much of the Operations Research described above was conducted at the very dawn of the computer revolution. Since then, a number of research efforts have contributed towards the development of *computerized* tactical decision aids which incorporate many of the principals and strategies developed through Operations Research.

The Naval Air Development Center, for example, has sponsored research efforts to model plan recognition agents that operate within adversarial domains. For program development, verification and validation purposes, Computer Adversarial Agents that model enemy platforms (e.g. aircraft and ships) are generated. These computer adversaries pose a threat to Naval aircraft carrier Task Forces and are capable of interaction in a dynamically changing world.

The *intelligently guided operators* of Azarewicz's plan recognition systems project plan hypothesis forward in time much like implementations of the minmax game principal. Due to the dynamics of the battlefield, however, Azarewicz's models use *differential* gaming principals that better model domains where joint moves by both opposing forces might simultaneously occur (Azarewicz, 1987).

3.1.3 Combat Commanders

The preceding section introduced the application of *game theory* to computer generated autonomous adversaries. This section will introduce the use of expert system technology as it is applied to model a combat vehicle commander.

Gibson describes an expert system used to model a combat vehicle commander's thought or combat decision-making process (Gibson, 1989). Additionally, Gibsons system applies MYCIN certainty factor methodology to model uncertainty common to many battlefield situations.

3.1.4 Adaptive Maneuvering Logic - 90

While the previous section introduced an expert system based vehicle commander, this section will describe a fully autonomous rule based air combat adversary that GreyStone Technology has commercialized.

Adaptive Maneuvering Logic - 90 (AML-90) is an advanced, synthetic adversary control model that allows for real-time, interactive air combat with a six degree-of-freedom air combat simulation for one-versus-one and two-versus-one engagements (GreyStone, 1994). The decision-making process is implemented using a rule-based system that contains a set of air combat rules and associated target behavior modes that consider multiple phases of a fighter combat mission including: Beyond Visual Range (BVR), Intercept, Close-in-Combat (ClC), and Bugout. The adversaries execute in real-time to provide realistic air target simulation for air engagements.

The GreyStone Adaptive Maneuvering Logic - 90 (AML-90) software provides several user selectable aerodynamic models of fighter aircraft platform and allows for 4 computer generated pairs (ownship and wingman), 8 aircraft total, to be simulated simultaneously during a session. The AML-90 pairs can be controlled as adversarial forces within a simulation exercise and may be directed by the user to engage other aircraft entities in either a 1-v-1 or a 2-v-1 engagement.

The Relative Reference Display (RRD) allows the user to control and monitor the AML-90 simulation environment. The RRD provides a two dimensional gods-eye-view of both the simulated arena and the CGF. A Graphical User Interface (GUI) is provided by the RRD which allows the user to set-up the initial positions of AML-90 entities and to establish routes of flight. The GUI also allows the user to specifically control the targets that the AML aircraft will intercept or engage.

3.2 Early Entertainment Environments

The previous section (3.1) offers examples of early applications of CGF technology to government simulation and decision aiding environments. The early government applications

clearly demonstrate that limited computer processing power forced a development emphasis on high fidelity Intelligent Adversaries. At the same time, however, commercial gaming applications were developed with an emphasis on high fidelity *presentation* because the limited computer processing power allowed for only rudimentary or brute *force* computer adversaries in the gaming environments.

The following sections offer examples of early applications of CGF technology to commercial gaming environments. GreyStone's *Purple Heart Corner* is exemplar of the presentation fidelity common to high end adversarial gaming environments. GreyStone's *Pteranodon Experience* is also an early example of the high fidelity presentation system common to many commercial gaming environments.

3.2.1 Purple Heart Corner

Purple Heart Corner is a commercially available entertainment game that combines state-of-the-art in virtual reality computer graphics with a detailed mock-up of a WWII bomber gun station. The shell of the gaming simulator closely represents the interior of the B-17 bomber at the waist gunner position, complete with stringers and bulkhead rings. An accurately-sized *window* opening in the fuselage side holds a copy of the 50 caliber Browning machine gun, and is flanked by a standard issue ammunition box (GreyStone, 1992).

The replica 50-cal Browning machine gun faithfully reproduces the heft and feel of the original gun, while a built-in pneumatic actuator recreates the hammering recoil of the big weapon. The sights are also accurately replicated, allowing the user to glimpse the skill required to aim the Browning accurately in the heat of the battle.

A description of the action of air combat experience shows that a heavier emphasis is placed on presentation rather than on the intelligence of the computer adversaries.

Me109's and FW190's, silhouetted against the sky, drop in from the south west, ahead of the formation. The relentless drone of the engines is interrupted, first by the top turret gunner, as he sends orange-red tracers out to greet the incoming fighters. Wave after wave of fighters dive through the formation. Tracers arc across the smoky sky. The 50-caliber Browning clatters on its mount as the fighters loom in the sights, with muzzles flashing. A hit! White flames claw the fighter to pieces as it spins downward, raining embers and trailing smoke. No time to exult; you turn to face the next attacker.

The above excerpt illustrates that a heavier emphasis is placed on graphics and sound technology. Though the experience provides computer generated targets as well as adversaries, they are controlled using scripted programming techniques.

3.2.2 Pteranodon

The Pteranodon experience, first developed as a showcase for Silicon Graphics' powerful Onyx image generator, represents the state of the art in premium virtual reality. The Pteranodon experience offers 180 degree visibility afforded by three large screens, and thousands of fully-textured, anti-aliased polygons refreshing the screens at thirty times a second. The detailed, colorful textures and realistic movement of objects in the simulation are complemented by the rich, booming, natural sounds of the environment (Crowe, B., 1994).

Although the Pteranodon is programmed to follow the commands of a rider, an intelligent obstacle avoidance system will execute a course deviation when necessary to avoid a collision with obstacles. The Pteranodon is also programmed to search for and follow other creatures with its gaze.

A description of the action of the Pteranodon experience shows that a heavier emphasis is placed on display and presentation.

Echoes of a primal screech rumble through the canyon, announcing the arrival of the raptor. Wings sweep the sky, mocking the winds. A rider guides the beast around cascades of water which plummet from dizzy heights to the river below ... vampire bats flutter above the next bend and monstrous wasps dive and swoop over a whirlpool that devours the river.

As the master of the Pteranodon, you guide it with the reins and by leaning in the saddle. It obeys your every command, but as you cruise through the canyons of this fantasy world, it skillfully avoids obstacles on its own.

The above excerpt should illustrate that a heavier emphasis is placed on graphics and sound and other presentation technology.

4. Recent Simulation and Gaming

The previous section (3.0) discussed early applications of CGF Technology and showed that limited computational resources forced government and commercial CGF applications to place differing emphasis on the fidelity of the Intelligent Adversary. This section will demonstrate that increased processing power has made it possible to incorporate both high fidelity CGF and high fidelity virtual reality in today's simulation and gaming environments.

4.1 Recent Government Simulation

A review of more recent application of CGF technology to government simulation environments will show a continued emphasis on high fidelity Intelligent Adversary technology but will also show a move towards the coupling of the technology with a high fidelity real-time virtual reality graphics environment.

The government simulation community has realized the full potential of today's computer technology and has coupled Computer Generated Forces technology with high fidelity real-time Virtual Reality. The Naval Postgraduate School's continued development of *NPSNET*, internationally known for its networked virtual environments technology, has incorporated Autonomous Players into their virtual battlefield. GreyStone Technology's *AML-D RAGE* , like NPSNET, is a networked application that combines the latest in real-time VR technology with high fidelity Computer Generated Forces technology. Both the NPSNET and AML-D RAGE applications will be discussed in more detail.

4.1.1 NPSNET Autonomous Players

The Navel Postgraduate School has included Autonomous Players in the NPSNET simulation environment to "provide interactive players when live players are not available or affordable (Zyda 1994)". The NPS Intelligent behaviors are modeled using expert system rule based technology capable of commanding unmanned vehicles in the simulation environment.

Among the autonomous NPS players are the autonomous tank forces players that provide intelligent behavior models which have the capability to work cooperatively. If numerous entities, for example, approach the autonomous tank force from several directions, then the NPS autonomous computer generated force can distribute their fire such that some tanks fire one direction and others in another direction. Should the autonomous player become outnumbered, it has the additional capability to call for reinforcements.

The NPSNET autonomous players are also using elevation data to reason about terrain. If, for example, a forward observation vehicle has Line of Site (LOS) with an enemy vehicle, the Autonomous Player can relay the coordinates to one of several howitzers. The threat is fired upon if it is in range of the howitzer.

Additionally, the autonomous players are equipped with intelligence reports that provide them with knowledge of the battlefield. Given whether a vehicle is friendly or not, the autonomous players can prioritize targets so that those targets with a higher priority are fired upon before lower priority vehicles.

4.1.2 RAGE AML-D

GreyStone's Real-time Advanced Graphics Environment (RAGE) coupled with AML-D is a showcase of both high fidelity graphics and intelligent adversaries running seemlessly together.

RAGE™ is a 3D visualization product designed for US and foreign government agencies and military services, and any members of the US or international defense industries who simulate operational scenarios, avionics and weapons systems, airframes, and mission planning/preview/rehearsal/training systems or conduct range operations for test or training, or manage C4I systems. It is particularly designed for organizations that have a need for advanced visualization but do not have the time, resources, or expertise to buy and build their own visualization products.

RAGE™ provides a 3D visualization component for avionics, weapons and aircraft system simulations, constructive and virtual mission simulations, mission preview, rehearsal and training systems/simulators, live training missions and actual combat missions. It can receive object (entity) state and event data from multiple intelligence and instrumentation sources and present a near real-time representation of live or simulated events at a 30Hz refresh rate. In addition to three standard viewpoint options:

Stealth, Out-the-Window, and Tethered, RAGE™ features two simultaneous viewpoints (i.e., Stealth on one monitor, out-the-window on another), sensor-related control functions including scan, slave, and zoom, and 3D visualization of non-visible phenomena such as weapons envelopes, platform signatures, EW signals, sensor beams and scan volumes, and operational area boundaries. RAGE™ can interface with large multi-screen displays, single screen systems, standard system monitors, helmet mounted displays and mini-domes (Shillicutt, 1994).

Depending on user requirements, RAGE™ can be integrated with multiple simulations, various user input/output interfaces, and display alternatives. Specific models and environment renderings can be produced. Non-visible phenomena functionality can be made dynamic such that they respond to the physical criteria which influence their behavior. Examples include dynamic SAM envelopes based on target altitude, and velocity vector and radar detection volumes based on pulse repetition interval or radar cross section.

Distributed Interactive Simulation provides a specialized method for integrating simulations into your visualization environment. RAGE™ is certified to receive DIS entity state and event PDUs. If a user needs to add a 3D visualization product to any entity state source (constructive or virtual simulation, live range data or live combat data), entity state source can be translated into DIS protocols.

AML-D is a (DIS) compliant version of the Adaptive Maneuvering Logic - 90 software (detailed in section 3.1.4 and GreyStone '93) that translates entity state data to DIS PDU which are forwarded to the RAGE application. This combination of high fidelity VR and CGF technology allows for interaction between dynamic AML-90 aircraft and large multi-player exercises (GreyStone, 1994).

4.2 Commercial Entertainment

The following sections will show that commercial entertainment has realized the full potential of today's computer technology and has incorporated intelligent adversary technology and high fidelity real-time virtual reality technology in a *single* synthetic environment.

The ThunderBolt commercial gaming environment, developed at GreyStone for a state-of-the art amusement park ride, is a synthetic environment that satisfies many requirements also common to government simulation. *Thunderbolt* has successfully combined high fidelity adversary technology with state of art presentation technology.

4.2.1 The Thunderbolt Experience

The computer generated forces developed for the ThunderBolt application are designed to provide a constant level of action for the human players who participate in the gaming environment. The underlying goal is to keep a continuous flow of adversary aircraft (both target and threat) in the field of view of each of the human players.

The technologies used to develop the ThunderBolt intelligent adversary fundamental behaviors are based on the modeling techniques utilized within the AML-90 adversary software. Although the number of actual CGF players required for the experience is significant, the constraints of the ThunderBolt compute environment allowed a CGF design based on a derivative of AML-90 behavioral model.

Like AML-90 adversaries, the ThunderBolt CGF derive relative threat geometry and apply a set of logic in order to assess an appropriate action. The logic is both phase and goal-based in that geometrical parameters such as range and target aspect are determined. The four CGF phases are *Intercept*, *Engage*, *CIC*, and *Bugout*. The phase is used to determine the set of tactical logic to apply to the situation and ultimately determines the CGF's flight behavior and actions. While the AML-90 CGF includes a robust set of tactical logic, including cooperative logic with a wingman, the ThunderBolt CGF operate independently of one another.

5. Conclusion

GreyStone Technology believes that many of the needs of both government simulation and commercial entertainment communities can be satisfied through Virtual Environments technology. Furthermore, these virtual environments are combinations of Multi-Sensory Human-Computer interfaces with real-time simulations that are populated by dynamic, intelligent and interactive behavioral models (CGF). In the final solution, the distinction between government CGF and commercial Intelligent Adversaries, is defined by the user's needs and the personality or *behavior* of the application. The underlying software structures and technologies should be common and

reusable. As the user determines the fidelity of both the adversaries and the interfaces needed, a compromise must be made on the requirements of costs and operational logistics.

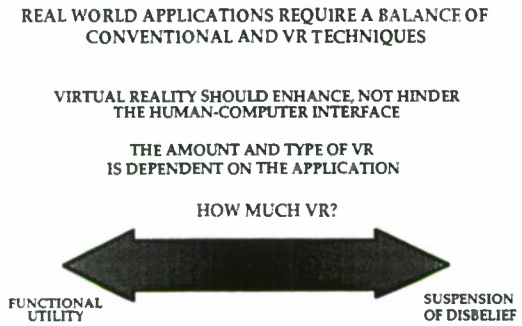


Figure 1: VR Application Axioms

As a single solution for all applications will unlikely satisfy all users, we propose that a common foundation class of object oriented CGF libraries can be cost effectively shared across both commercial entertainment and government simulation applications. With the axioms shown in figure 1 above, the end user can determine the optimal operations point and the developer can determine which of the libraries are needed to ensure the requirements of a specific exercise or experience are met with optimal efficiency.

6. References

Azarewicz, J., et. al. (1987) "Multi-Agent Plan Recognition in an Adversarial Domain", *Third Annual Expert Systems in Government Proceedings*, pages 188-193, Washington, DC, October, 1987.

Crowe, B., "Pteranodon Sighting at SIGGRAPH '93", *Virtual Reality World*, November 1994

Crowe, M., "Virtual Environments at GreyStone", *technical presentation*, GreyStone Technology

Gibson, T. J., "Modelling a Combat Vehicle Commander with an Expert System", *DTIC*, AD-A208 533, 1989.

GreyStone (1994), "AML-D User's Manual", *GSD-AMLD-UM110*, GreyStone Technology.

GreyStone (1992), "Purple Heart Corner", *Tech Memo*, GreyStone Technology.

GreyStone (1993), "ThunderBolt", *Tech Memo*, GreyStone Technology

IST, "A Glossary of Modeling and Simulation Terms for Distributed Interactive Simulation" *11th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, Vol. 1, 1994

Morse, P.M., Kimball, G. E. (1951) Methods of Operations Research, First Edition, Peninsula Publishing.

Shillicutt, D., "On the Cover ...", *Simulation*, Vol. 63, No. 5, 1994

Von Neumann, J. and Morgenstern, O. (1944) Theory of Games and Economic Behavior, Princeton University Press.

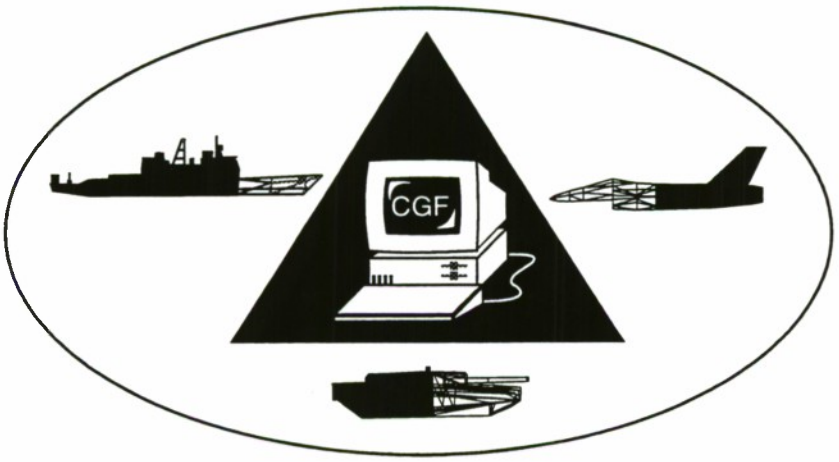
Zyda et. al., "The Software Required for the Computer Generation of Virtual Environments", *Presence*, Vol. 2, No. 2.

7. Authors' Biographies

Rich Warren is Staff Engineer and Intelligent Systems Technology Group Leader at GreyStone. Mr. Warren holds a Bachelors degree in Computer/Cognitive Science. His research interest is in Artificial Intelligence and Autonomous Intelligent Adversary's.

Mike Crowe is the Director of Technology at GreyStone. Mr. Crowe holds a Bachelors degree in Cognitive Science/Artificial Intelligence. His technical focus is on the research and development of advanced, automated control, and decision-making methodologies to provide intelligent computer-generated players within virtual environments.

Don Shillcutt is Principal Systems Engineer at GreyStone Technology. Mr. Shillcutt holds Masters degrees in Electrical Engineering and Management. His principal area of interest is in Information Technology.


CGF

CGF Opportunities in Plowshares

Mikel D. Petty and Mary P. Slepow
Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826-0544
mpetty@ist.ucf.edu cslepow@ist.ucf.edu

Paul D. West
Unites States Military Academy
West Point, NY 10996
fp8049@trotter.usma.edu

1. Abstract

Plowshares is a military sponsored project to apply military constructive simulation technology to training and analysis for emergency management. In the first phase of the Plowshares project, the U. S. Army's Janus simulation model is being modified and extended to support certain types of emergency management scenarios (hurricanes and chemical spills. In order to be useful, emergency management simulation will require CGF capabilities. The capabilities will be similar, but not identical, to the CGF capabilities already developed for battlefield simulation.

2. Plowshares

2.1 Project Overview

The Plowshares project is a military sponsored effort to apply military constructive simulation technology to training and analysis for emergency management.

In the first phase of the project the project team members are collaborating on an effort which will culminate in a "Proof of Principle Demonstration" scheduled for August 1995. For that demonstration, the project team will be modifying and enhancing the U. S. Army's Janus simulation model (described below) to permit it to support two types of emergency management scenarios: a hurricane and a chemical spill. Both of those scenarios will be located in Orange County Florida.

2.2 Project Organization

The Plowshares project team is composed of four organizations; each has specific project functions to perform They are:

1. U.S. Army Simulation, Training, and Instrumentation Command (STRICOM)
- Project management
- Janus software modifications

2. United States Military Academy at West Point (USMA)
- Terrain generation
- Scenario generation
- Janus software modifications

3. Training and Simulation Technology Consortium (TSTC)
- Requirements analysis
- Commercialization and marketing

4. Institute for Simulation and Training (IST)
- System integration and testing
- Model survey
- Documentation

In addition to the four organizations listed above, Orange County Florida is also playing a key role in the project. County personnel are providing considerable time and information in support of the requirements analysis process. Furthermore, it is planned that county personnel will operate the Plowshares simulation during the Proof of Principle demonstration; that topic will be discussed below.

2.3 Project Goals

The primary goal of the project is to implement a computer simulation that can simulate natural and man-made disasters and the actions taken in response to them. The simulation is intended to serve the purpose of training local authorities.

Implicit tasks associated with that goal are to:

1. Convert a military training simulation to a civil emergency management application.
2. Conduct a Proof of Principle Demonstration of that simulation.
3. Show reuse capability to the sponsor (U.S. Army).
4. Serve as a technology transfer initiative.

2.4 Using the Simulation

Once delivered, the Plowshares simulation will be used to train local authorities at the emergency management decision maker level. The simulation will be used in a manner known as a "command post exercise" in military training practice. While being trained, persons in charge of fire and rescue departments, police departments, public works departments, and so on will receive information, make decisions, and give orders in their usual manner, operating from their normal emergency operations center. However, the source of the information they receive will not be an actual emergency; rather, it will be the Plowshares simulation software running the training scenario. Similarly, their orders will not go to actual fire fighters, police officers, and medical personnel for execution; instead they will be input into the simulation, which will determine the results of the decisions. The persons actually operating the simulation will be emergency management personnel who have been trained to use the software so that the orders given by the decision makers can be properly interpreted by subject matter experts.

2.5 Introduction to Janus

Janus is an "interactive, two-sided, closed, stochastic, ground combat simulation" (Titan,1993). It is used by military analysts, who control the actions of simulated combat entities during execution, for analytical purposes including evaluating new weapons systems, tactics, and force structures. The primary focus of the Janus simulation is on ground combat maneuver and artillery units. Janus typically simulates individual vehicles and infantrymen, tracking their movement across the terrain and resolving combat at the level of the individual entity; groups of entities can also be represented. Combat, such as direct fire, is resolved stochastically.

Janus models weather and its effects, day and night visibility, engineer support, minefields, aircraft, and chemical weapons. Janus uses a digitized terrain database format developed by the Defense Mapping Agency that can represent contours, roads, rivers, vegetation, and urban areas. Terrain affects movement and visibility in a realistic manner. Janus typically runs on workstations and supports scenarios of up to battalion size.

3. Plowshares Test Demonstration

3.1 Test Demonstration Overview

Prior to the full-scale implementation of Plowshares, a preliminary test of the adaptability of the Janus simulation to emergency management simulation was performed. In this Plowshares test demonstration, STRICOM and USMA personnel made a moderate set of modifications to the Janus "databases", or configuration files, to adapt it to a simple hurricane scenario.

The test demonstration scenario proceeds as follows. A hurricane under the control of the simulation moves through "Tuskawilla", an actual residential subdivision in Seminole County Florida, which was recreated in the Janus terrain database format. The hurricane has various effects on the area, including creating rubble, starting fires, knocking out power stations, and triggering a small chemical spill.

Once the hurricane itself has cleared the simulation area, the scenario continues. Citizens flee the hurricane under the control of the computer simulation software. Police, fire, and public works vehicles and personnel respond to the storms effects.

For the test demonstration, Janus' existing capabilities were used as much as possible. Those existing capabilities are combat oriented. Therefore the hurricane was modeled as a formation of slow moving helicopters and its effects on the buildings were modeled as weapons firing. Similarly, the fire trucks move as if they were military vehicles and put out fires with a different form of weapons firing.

3.2 Terrain and Systems Modeling

In Janus, the term "system" refers to a single simulated entity, such as a vehicle or person. This section and the following one will adhere to that usage.

Modeled Emergency Management System	Original Janus Military System	Comments
Hurricane*	Mi-24 Hind helicopter	8 Hinds in fixed formation
Police car	High-Moblity Multipurpose Wheeled Vehicle (HMMWV)	Higher maximum road speed
Police officer	Infantryman	
Bulldozer*	D-7 military bulldozer	Equipped with M-16, used to clear rubble
Dump truck	2 1/2 ton truck	
Fire truck	M-106 self-propelled mortar carrier	
Medical evacuation helicopter	UH-60 Blackhawk helicopter	
Ambulance	HMMWV ambulance	

Table 1. System mappings in the Plowshares test demonstration

The basic Janus terrain database was overlaid with three terrain zoning categories. Their characteristics affected movement and lines of sight. These categories were:

1. Single-story basic residential
2. Two-story commercial.
3. Open areas (e.g. parks and clear spaces); these were used as areas in which to locate schools, etc.

Significant individual buildings and building types were modeled as Janus systems using existing characteristics from the Janus database. Systems (i.e. vehicles) were used because Janus does not represent individual buildings. This was done by finding the best match between an existing Janus system and the desired building type. The Janus database entries then were modified to represent the building type as well as possible. The result of this process was that various armored vehicles were transformed into buildings. This required deleting their weapons and ability to move and increasing their size and personnel-carrying capacity.

In Janus, "enemy" systems only appear on "friendly" screens when there is a direct line of sight with a friendly system. In the Tuskawilla hurricane scenario, this means that for the hurricane, fires, smoke, and so on (enemy systems) to be visible on the screen, one of the police cars, fire trucks, and so on (friendly systems) had to have a line of sight to them. Because we wanted the hurricane and its effects to be continuously visible for the demonstration, the buildings were given the ability to "see" for short distances. Additionally, invisible systems with no screen display icon were positioned to ensure that the storm and its effects would be seen.

Representing emergency management vehicles and personnel (the friendly systems) was easier; existing Janus systems were modified (in the database) to match their characteristics. Table 1 summarizes those changes. In all these cases, new screen display icons were drawn to represent the demilitarized system.

3.3 Events Modeling

The hurricane's movement was scripted so that representative effects would occur time synchronized with its passage. These effects took five forms:

1. Off-screen artillery fired smoke rounds onto burning buildings to provide signatures.
2. Off-screen artillery fired high explosive rounds to indicate rubbling.
3. Hidden fire icons moved onto burning buildings to enhance visual representation. These appeared on-screen when they were "seen" by the buildings.
4. Hidden rubble icons moved into scripted positions and appeared when seen.
5. A hidden spill icon began moving toward the river. Its speed was set so that a prompt response would stop it before it reached the river. It if reached the river the spill would spread downstream.

There were two interactions between emergency crews and storm effects:

1. The bulldozer cleared the rubble blocking a road by killing it with its M-16. This was the only direct fire event.

Event	How modeled	Comments
Storm moves through*	Grouped helicopters fly preplanned route	8 Hinds in fixed formation
Storm is continually observed*	Invisible spotters watch storm path	
Rubbling appears*	(1) Indirect fire (HE) from off-screen artillery (2) Hidden rubble icons move into position and are seen	
Fires appear at multiple sites*	Hidden fire icons move into position and are seen	
Smoke appears from fires	Indirect fire (smoke) from off-screen artillery	
Emergency crews respond	Preplanned routes for modified military vehicles	
Crews wait for rubble to clear*	Timed stop nodes on preplanned routes	
Dozer clears rubble blocking road	Rubble killed by direct fire, disappears from screen	Failure to clear possible (hit probability)
Spill approaches waterway	Timed, preplanned route for vehicle simulating spill	
Police cars dismount officers	Infantrymen dismount from vehicle	
Fire trucks put out fires	Indirect fire weapons kill fire	Fire truck and fire are both Janus systems

Table 2. Events in the Plowshares test demonstration

2. Fire trucks fired a combination of smoke and high explosive rounds (indirect fire) onto the burning buildings. This accomplished two things: the smoke continued to give the fire signature, and the high explosive rounds with a modified (low) probability of kill enabled the fire trucks to eventually kill the fire.

The events of the test demonstration scenario and how they were modeled are summarized in Table 2.

For the test demonstration, emergency crew reactions (i.e. movement paths, destinations, and actions) were also scripted. They were scripted for the test demonstration only so that the demonstration could run without intervention. Normally, the emergency crew reactions would be under the control (direct or indirect) of the simulation user or trainee, who would be learning how best to react, in terms of resource allocation and response timing, to an emergency and its immediate aftereffects.

3.4 Comments

The implementation of the test demonstration has proven to be a valuable exercise. It served to reveal in Janus both its flexibility and its limitations when applied without code modification to emergency management simulation. The system and event mappings shown in the tables that seem most forced, as indicated by an asterisk (*), suggest areas where special attention is needed.

4. Future Project Plans

4.1 Proof of Principle Demonstration

Janus was used essentially unchanged for the test demonstration; the scenario was implemented entirely within the Janus databases. The scenario scope and realism goals for the Proof of Principle Demonstration will require that much more significant changes be made to Janus. That work will be a major focus of the project team's efforts over the next eight months. Janus developers at USMA will be working to modify Janus with new capabilities needed for emergency management. These include

more realistic hurricane models and damage assessment procedures.

In August 1995 a Proof of Principle Demonstration of the enhanced Janus simulation (i.e. the Plowshares simulation) will be conducted at the Orange County Florida Emergency Operations Center. Two scenarios, a hurricane and a chemical spill, will be run on the Plowshares simulation. Orange County personnel will interact with the simulation and report the status of the emergency to high-level emergency response managers. Those managers will make decisions and give orders intended to save lives and property. Those orders will be interpreted and input into the simulation by their subordinates, where their effects will be determined by the simulation. Both exercises will be recorded and evaluated to determine the success of the development effort.

4.2 Model Survey

Janus was chosen as the base software for the Proof of Principle Demonstration because of its availability and seeming applicability. However, no final decision has been made that Janus is the best choice for long-term development of emergency management simulation. While Janus is being enhanced for the Proof of Principle Demonstration, a parallel task will be conducted by IST to survey the available set of military simulation models. Janus and a number of other models will be examined carefully and one selected for further development. Even if the initial selection of Janus proves to be the correct choice, surveying the other models will doubtlessly supply a wealth of ideas for enhancements and functionality for future emergency management simulation.

4.3 DIS Compatibility

In the first phase, Plowshares will run as a stand-alone constructive model. As might be expected given the project's military sponsorship, we will be examining the possibility of interconnecting Plowshares with virtual simulation, i.e. Distributed Interactive Simulation (DIS). Such a connection would allow low-level participation by trainees in virtual simulators (e.g. medical evacuation helicopters) in the same emergency response action that is being overseen by high-level response managers. The applicability of DIS, in both its current and future versions, to emergency management simulation in general is discussed in a companion paper (Loper,1995). Janus has already been linked experimentally to DIS (Pratt,1994), and some of that technology will be useful to bringing Plowshares into DIS.

5. Computer Generated Forces

5.1 Overview

The goal of the first phase of the Plowshares project is to show that Emergency Management is a valid application of constructive (and later, virtual) simulation technology. Emergency Management (EM) simulation will need computer controlled autonomous entities, i.e. Computer Generated Forces (CGF), for reasons and in roles analogous to those of CGF in battlefield simulation. This section will list some of those reasons and roles, identify some specific CGF-type capabilities needed in EM simulation, and compare and contrast those EM CGF requirements with what has been developed for battlefield simulation. We will move freely between CGF capabilities that relate to EM simulation in general and those that relate to Plowshares in particular.

5.2 CGF roles in EM simulation

There are two roles for CGF entities in EM simulation: low-level entity control and high-level replacement players. Each will be discussed in turn.

5.2.1 Low-level entity control

As described previously, the goal of the Plowshares simulation is to train EM managers. Those individuals typically make resource allocation decisions at a high-level and rarely exercise control over the emergency response units at a detail level (such as planning a route for a fire truck). However, in the current Plowshares simulation many of the low-level behaviors must be controlled by a human operator; for example, detailed routes must be planned by the operator for vehicles. Many other low-level behaviors are also under operator control. This characteristic is inherited from the Janus software base.

The result is that human operators, who are part of the simulation rather than trainees, are required to use the Plowshares software in a training exercise. These operators must interpret the command decisions of the trainees and give detailed low-level behavioral commands to the simulation entities. Obviously, this raises the expense of an exercise and reduces the

flexibility of the software. CGF capabilities are needed to provide low-level EM behavior control for the entities. The degree to which low-level EM behaviors resemble battlefield behaviors varies by behavior.

5.2.2 High-level replacement players

EM simulation will also need CGF replacement players for high-level decision makers. In the context of a Plowshares-style county-wide scenario, such decision makers might be the chiefs of the typical EM response departments:

1. Fire and Rescue
2. Public Utilities
3. Public Works
4. Health Services

In a battlefield simulation, these decision makers might be analogous to battalion or brigade commanders. For the most part, CGF systems for battlefield simulation have to date not provided decision makers at that level (although the ongoing ARPA Command Forces project is intending to do so). Their omission has not been a severe limitation for two reasons. First, useful training can be conducted with lower-level scenarios, such as company-sized actions. Second, human controllers have been available to provide higher-level decision making where needed.

Unfortunately, these two reasons are less applicable to EM simulation. First, virtually all interesting EM training will require more than one of the response departments. For example, there are not many "police only" emergencies; even something like a riot will involve medical services to evacuate and treat casualties and fire and rescue services to fight fires that may start. The same comment applies to the other branches. Second, it will be more difficult to provide human operators to fill in for decision makers not available for the exercise because of the wide differences in required expertise and inter-departmental separation. Consequently, EM simulation will need CGF replacements for these decision makers early in its development to provide training flexibility, e.g. so that the Police department can have a meaningful exercise even if the Fire department is not available to participate.

5.3 CGF capabilities in EM simulation

How applicable are CGF capabilities already developed for battlefield simulation to EM simulation? This section will examine a few common CGF functions and compare and contrast them in the CGF and EM contexts.

5.3.1 Route planning

The basic CGF capability is route planning. In its simplest form, a CGF entity is able to move from its current location to an assigned destination after autonomously planning a route between the two points. In battlefield CGF systems, the CGF system's route planner might consider factors such as distance, terrain trafficability, unit boundaries, and cover and concealment from enemy fire. In EM simulation, there are three interesting aspects of route planning.

First, the terrain, and thus the terrain database (TDB), is likely to be very different in "feel" from a typical battlefield TDB. In particular, EM TDBs will often be primarily or entirely urban. That means that they will be dense with features, such as buildings, walls, bridges, and roads of significantly different types. There will also likely be "zones", or regions of the TDB defined to have specific characteristics (e.g. residential or light industrial) that will also influence route planning.

Second, factors other than the TDB itself will affect EM route planning to a degree greater than battlefield route planing. One such factor is road blockage. Often the emergency itself (hurricane, earthquake, or flood) will block roads or other movement corridors; this might be a form of dynamic terrain. A route planning algorithm should consider blockages only if the fact of the blockage would be known to the entity for which a route is being planned. A fire truck that could not be aware that a building has collapsed onto an important access road should not route around that point; instead, it should plan a route through that point, and when the blockage is encountered, the route should be replanned (and the blockage reported). Roads may also be blocked by masses of citizens fleeing a disaster; however it may be valid to allow for that type of blockage prior to discovery under the assumption that the EM entities would expect it.

Finally, there is to some degree an assumption of optimality implicit in route planning by EM entities. The reason for this is that, unlike combat entities, EM entities are crewed by personnel who are typically very familiar with the terrain in which they are operating; the driver of a police vehicle normally knows his or her precinct very well. Therefore,

within the limits of available information, an EM route planning algorithm should normally find minimum time routes. That level of performance may not be demanded from battlefield CGF systems.

5.3.2 Cooperative behaviors

The repertoire of behaviors expected from EM entities include many that are cooperative at a low-level, possible more than in battlefield simulation. A few examples should suggest the set:

1. A bulldozer and a dump truck cooperate to clear rubble
2. Multiple police officers dismount from a vehicle and disperse to separate intersections to direct traffic
3. A fire truck waits for a bulldozer to clear rubble, rather than rerouting around it, based on travel time estimates

These low-level entity-to-entity cooperative behaviors will have to be provided in an EM CGF application. Of course, high-level cooperation between groups of EM entities is the responsibility of the human trainees, and a training goal of the simulation.

5.3.3 Citizens

Realistic EM simulations will need to include citizens because of the many ways they affect the response to an emergency. Citizens flee the disaster, often in non-optimal ways, and clog the road network, requiring avoidance in route planning and traffic direction. Citizens are injured, thus requiring medical assistance from medical entities. Citizens take advantage of the chaos associated with an emergency and loot residences and businesses, and therefore constitute a problem for the police entities to solve. Providing all of these behaviors in an EM application will be required. Furthermore, the EM CGF system will have to control the citizens essentially without operator intervention because their numbers will likely be too large to do otherwise.

6. Conclusions

Simulation is growing in importance as a tool to prepare and train for emergencies. The U. S. military's responsibilities in emergency response are expanding. The Plowshares project team intends to apply the large body of military simulation technology to emergency management simulation.

To be useful, emergency management simulation will require CGF capabilities that are similar but not identical to those developed for battlefield simulation. The extent to which battlefield CGF software and algorithms can be transferred to the EM domain remains to be determined.

7. Acknowledgment

The preparation of this paper was supported by the U. S. Army Simulation, Training, and Instrumentation Command under the Plowshares project, contract N61339-95-K-0003. That support is gratefully acknowledged. However, all comments in this paper are the responsibility of the authors and do not necessarily reflect STRICOM positions.

8. References

Loper, M. L. and Petty, M. D. (1995). "Distributed Interactive Simulation and Emergency Management", *Proceedings of the 1995 SCS Simulation MultiConference*, Simulation for Emergency Management, Society for Computer Simulation, Phoenix AZ, April 9-13 1995.

Pratt, D. R., Johnson, M., and Locke, J. (1994). "The Janus/BDS-D Linkage Project: Constructive and Virtual Model Interface", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 4-6 1994, pp. 443-448

Titan, Inc. (1993). *The Janus 3.X/UNIX Model User's Manual*, W800XR-3125-0052, TRADOC Analysis Center.

9. Author's Biographies

Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He is currently managing Plowshares, an emergency management simulation project. Previously he led IST's Computer Generated Forces research projects. Mr. Petty received a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from California State University, Sacramento. He is currently a Ph.D. student in Computer Science at UCF. His research interests are in simulation and artificial intelligence.

Mary P. Slepow is an Engineering Writer for the Plowshares project at the Institute for Simulation and Training. She has a B.S. in Mechanical Engineering from the University of Florida and has experience in

the power generation field. Ms. Slepow is currently a graduate student in Instructional Technology at the University of Central Florida.

Paul D. West is the Warfighting Simulation Manager in the Department of Systems Engineering at the United States Military Academy in West Point NY. He received an MBA in Management from Long Island University and a B.S. in Liberal Arts from the University of the State of New York. Mr. West has an extensive military background and is a former Assistant Professor of Military Arts and Science at West Point.

Planning for Reactive Behaviors in Hide and Seek*

Michael B. Moore, Christopher Geib, Barry D. Reich

University of Pennsylvania
Department of Computer and Information Science
200 S. 33rd Street, Philadelphia, PA 19104-6389

March 16, 1995

1 Abstract

We describe the ZAROFF system, a plan-based controller for the players in a game of hide and seek. The system features visually realistic human figure animation including realistic human locomotion. We discuss the planner's interaction with a changing environment to which it has only limited perceptual access. A hierarchical planner translates the game's goals of finding hiding players into locomotion goals, assisted by a special-purpose search planner. We describe a system of parallel finite state machines for controlling the player's locomotion. Neither path-planning nor explicit instructions are used to drive locomotion; agent control and apparent complexity are the result of the interaction of a few relatively simple behaviors with a complex (and changing) environment.

2 Introduction

The game of *hide and seek* challenges the ability of players to plan for acquiring information and to react quickly to what they see. The player who is "it" (the *seeker*) must explore his environment attempting to locate other players (*hiders*). Those players must select hiding places which are difficult to discover while providing access for them to run safely to home base when the way is clear. The goal of this work is to develop simulated agents that can play hide and seek (or more dangerous games) (Moore, Geib, & Reich 1995).

We describe a planning system for a player (which can change roles between hider and seeker during the game) and its vertical integration into a system called ZAROFF that selects reactive behaviors to execute in an animated simulation. Operation of the planner is interleaved with execution of the reactive behaviors so that the agent may adapt to a dynamic environment.

The software chosen for this work is *Jack*® (Badler, Phillips, & Webber 1993) running on Silicon Graphics workstations. *Jack* is a human modeling and simulation program developed at the Center for Human Modeling and Simulation at the University of Pennsylvania, that features visually realistic human locomotion based on both kinematic and dynamic techniques (Ko 1994; Ko *et al.* 1994). *Jack*'s LISP application programming interface (Becket 1994) was used to implement ZAROFF. This interface supports access to the environment (a database) and its behavioral simulation system.

3 Generating Behaviors

Human locomotion is performed by the Behavioral Simulation System (BSS) (Badler, Phillips, & Webber 1993; Becket & Badler 1993). ZAROFF controls this locomotion indirectly by binding behaviors to human figures in the environment database. BSS, which constantly monitors the environment, immediately initiates locomotion based on the agent's bound behaviors.

A player utilizes a set of behaviors in interacting with its environment. ZAROFF includes the following behaviors: attraction, avoidance, field-of-view (to avoid areas visible to the seeker), path follow-

*This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

ing (to draw an agent to a path) and chasing.

4 Action Execution

The Action Execution module is responsible for the control of all actions occurring in ZAROFF. Most actions such as opening and closing doors are performed directly by this module. (As noted earlier, human locomotion is performed by the Behavioral Simulation System (BSS) (Badler, Phillips, & Webber 1993; Becket & Badler 1993), so is controlled only indirectly by Action Execution.)

Non-locomotion actions are performed directly by Action Execution manipulating the environment. For example, a door is opened by rotating it about its hinges. This rotation is done incrementally, a small amount each frame of animation.

Locomotion is performed indirectly by Action Execution binding behaviors to human figures in the environment database, which means neither path-planning nor explicit instructions are used to drive locomotion: rather, agent control and apparent complexity are the result of the interaction of a few relatively simple behaviors with a complex (and changing) environment. An agent is made aware of its environment through the use of a network of sensors. Based on the information gathered by these sensors the path through the terrain is incrementally computed. This allows the agent to react to unexpected events such as moving obstacles, changing terrain, or a moving goal (Reich *et al.* 1994).

4.1 Sensors and Behaviors

A *sensor* senses an object or location in the environment. It is a function which returns the distance and angle to that object or location relative to the agent's position and orientation (his state). A *control behavior*, such as *attract* or *avoid*, is a function that maps distance and angle to a stress value, where lower values represent more desirable states. The combination of a sensor and a control behavior results is referred to as a *behavior*. The agent utilizes a set of behaviors in interacting with its environment. The following three classes of behaviors are currently in use.

Attraction: An attraction behavior draws an agent toward a goal – either an object or a location. If a goal object moves, the point of attraction moves appropriately. The output (stress value) of an attraction behavior is high when the agent is far from the goal and decreases as the agent nears the goal.

Avoidance: An avoidance behavior is used to avoid collisions between the agent and objects or other agents. The sensor component of an avoidance behavior has a sector-shaped region of sensitivity. If there are no objects in this region, the output of the avoidance behavior is zero. Otherwise the output is proportional to the distance and size of the detected objects.

Field-of-View: A field-of-view behavior uses a sensor to determine whether or not the agent is visible to any other agents. The output is proportional to the number of agents' fields-of-view it is in and inversely proportional to the distances to these agents. This behavior is primarily used to support hiders in ZAROFF.

4.2 The Behavioral Simulation System

BSS provides general locomotion of objects in *Jack* (Becket & Badler 1993), and is used in ZAROFF to generate human locomotion. The central control mechanism of BSS is a perception control, and action loop. During the perception phase the behavior outputs are determined, during the control phase the next foot position is selected, and during the action phase the step is taken.

4.3 Behavior Scheduling

For control of the agents' behaviors we use a set of finite state **machines** which run in parallel with the simulation. These machines are responsible for behavior scheduling and act as a high level interface to the behaviors. They may instantiate other machines and either run in parallel with them or wait for them to exit. Communication among state machines is also possible. We make use of this communication among machines to implement interaction between players (Section 5).

When the planner commits to an action it invokes the Action Execution module. The Action Execution module instantiates and runs the machines necessary to carry out that action. Each machine is responsible for the scheduling and control of the appropriate behaviors.

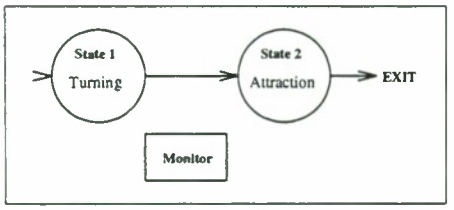


Figure 1: The Attract Machine

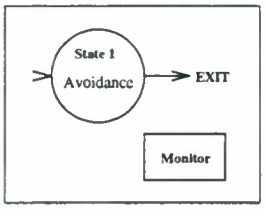


Figure 2: The Avoid Machine

4.3.1 Attraction

An *attract machine* consists of two states (Figure 1). In **state 1** the agent turns to face the object or point of attraction. In **state 2** an attraction behavior is bound to the agent. The agent walks to the goal object or location. When the agent arrives, this behavior is unbound and the machine exits.

An attract machine optionally includes a monitoring process which checks, in parallel, for an arbitrary condition (a LISP expression passed as an argument when the machine is created) to be true. If this condition evaluates to true at any time, the attraction behavior is unbound and the machine exits. The value returned by the machine when it exits indicates the cause of the exit.

We use the monitoring process primarily to control the behavior of the seeker. As the seeker explores the environment, moving from one location to another through the binding and unbinding of attraction behaviors, its progress is interrupted if at any time a hider becomes visible. This allows the seeker to stop and formulate a new plan, taking advantage of the opportunity.

4.3.2 Avoidance

An *avoid machine* consists of only one state (Figure 2). It binds an avoidance behavior to the agent for each avoidable object in the environment, and then exits. Unlike attraction, avoidance is maintained throughout the entire simulation. In ZAROFF the objects to avoid are doors, walls, and other agents. An avoid machine is used during the initialization of the simulation, once for each agent. This prevents the agents from colliding with each other, walls, and doors, and can easily be extended to any other objects that might be added to the system.

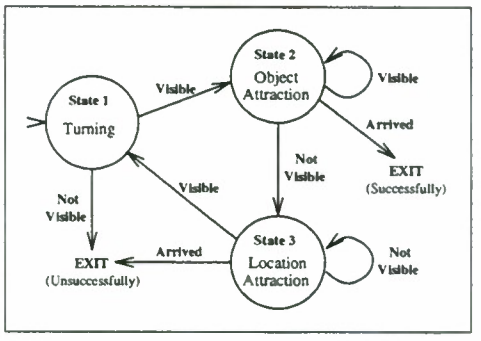


Figure 3: The Chase Machine

4.3.3 Chasing

The internal structure of a *chase machine* is a loop consisting of three states (Figure 3). If the target, the agent being chased, is not immediately visible, the machine exits unsuccessfully. Success is indicated by the exit value.

When the target is visible to the agent, the machine starts in **state 1**. The agent turns to face the target and the transition to **state 2** is made. An attraction to the target is bound. The agent walks toward the target until one of two things happen. Either the agent arrives at the target in which case the machine exits successfully, or the target ceases to be visible to the agent. The target may have walked around a corner. In the latter case the transition to **state 3** is made. An attraction to the last known location of the target is bound to the agent. The agent walks toward this location until it arrives or the target becomes visible again. In the former case the machine exits unsuccessfully. In the latter case the transition to **state 1** is made.

A potential improvement to this machine would be to add an Extrapolation State. If the machine is in **state 3** and the agent arrives at the last known location of the target without seeing the target, instead of exiting unsuccessfully the agent would walk in the direction the target was walking before disappearing. This would require maintain-

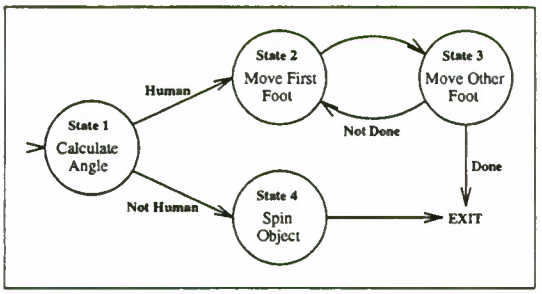


Figure 4: The Turn Machine

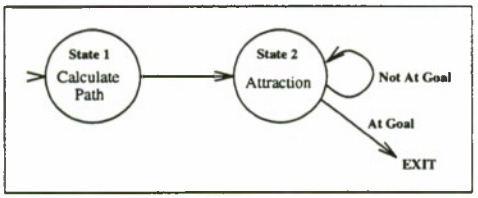


Figure 5: The Path Machine

ing two pieces of information about the target: its last known location and its vector velocity.

4.3.4 Turning

A *turn machine* is used to rotate an object or human to face an object or location. The rotation may be either clockwise or counterclockwise, whichever is smaller, or may be specified. A turn machine consists of four states (Figure 4). In **state 1** the angle and direction of rotation are calculated. If the object is not a human, it is spun about its center of mass and the turn machine exits.

Humans are a special case. The human agent rotates the first foot to the right for clockwise, or to the left for counterclockwise. The leg and body follow, maintaining the agent's balance. The rotation angle for the first foot is the minimum of the goal angle and ninety degrees. The agent then follows the first foot with its other foot. If the goal angle is greater than ninety degrees, this cycle repeats until the goal has been reached. An optional argument can specify that the agent not follow with the other foot in the final cycle. When locomotion is to follow turning, it looks more natural when the final cycle is not completed.

4.3.5 Path Following

The idea behind a *path following machine* is to generate a path from the agent's current location to a goal location somewhere in the world and have the agent follow the path to the goal. A path following machine consists of two states (Figure 5). In **state 1** the path is calculated as a series of connected segments. The path has three important properties: it connects the agent to the goal, it avoids obstacles by a clearance specified by the user, and it is a path of minimal length given the first two constraints. In **state 2** the agent is attracted to successive vertices along the path. As the agent approaches a vertex, the attraction to that vertex is unbound and a new attraction to the next vertex along the path is bound. When the agent arrives at the goal, the path following machine exits.

Our path following differs from path planning because the agent is not constrained to the path. Instead, an attraction to the path is bound to the agent. This behavior competes with any other bound behavior for control of the agent. If a moving object crossed the path, the path-following agent would avoid it, and if a hider came into view the seeker would pursue it.

5 Planning

A player's goals change during the course of a game of hide and seek. The seeker must first locate a hider and then tag it before it reaches home base. Hiders first attempt to locate a hiding place. When one have successfully hidden, it stays there until it is necessary to move. When flushed from their hiding place by the seeker, they attempt to move back to home base. If they successfully arrive home, they wait until the start of the next game when their behavior resets. If one is caught, the first one caught adopts the behavior of the seeker.

These high-level goals are quickly translated into situationally appropriate reactive behaviors by IT-PLANS (Geib 1992), a hierarchical planner. IT-PLANS interleaves hierarchical expansion with action execution. It does this by using an incremental left to right expansion of the frontier of the plan structure to successively lower levels of abstraction. Thus planning only takes place to the degree necessary to determine the next action to be carried out. This is important for this domain since the seeker does not have complete knowledge of the domain and its knowledge is constantly

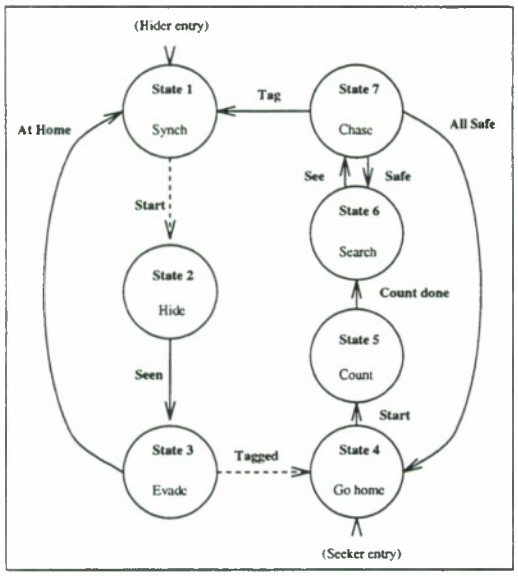


Figure 6: Player goal machine

changing.

A finite state machine is used to change a player's goals over the course of the game (Figure 6) and thus indirectly, what actions the player takes at different times during the game to achieve its goals.

The left side of Figure 6 corresponds to the hider role and the right side corresponds to the seeker role. There are two entry points to this network depending on which role the player will take initially; hiders start in **state 1** and seekers start in **state 4**. The hider starts hiding when the seeker begins to count. This is represented in the diagram by the dashed line. The transition in to **state 2** is in response to communication transmitted by the seeker's goal machine. Similarly, the transition from hider role to seeker role is in response to an external signal from the seeker's machine, being tagged.

When a player locates a hiding place (**state 2**), it remains there until seen by the seeker. Once discovered, the player attempts to evade the seeker and return safely to home base. Evading (**state 3**) is accomplished by combining avoidance of the seeker and attraction to base.

The seeker begins by going to home base (**state 4**) and synchronizing the start of the game by forcing the other players to transition from **state 1** to **state 2**. The seeker delays the start of its search by counting **state 5** to permit the hiders time to reach their hiding places. Then the seeker

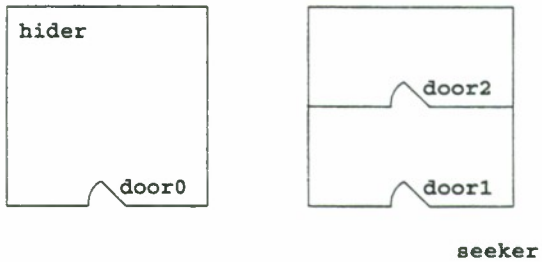


Figure 8: Plan view of example environment

begins to explore the environment to find a hider (**state 6**). When a hider is discovered, it becomes the target for a chase (**state 7**). The outcome of the chase determines whether the player will be a hider next game (when a hider is tagged before reaching base) or a seeker again (when all the hiders safely evade the seeker). If a single hider successfully evades the seeker, the seeker continues to search for other hiders until they are all safe. This may result in different hiders being chased at different times.

A consequence of limited perception is the occasional need to find objects. Our approach is to isolate this reasoning in a specialized module, a *search planner* that translates information acquisition goals to high-level physical goals to explore parts of the environment. Our approach to search planning requires that each player maintain information about the state of a heuristic search on an internal map. The heuristic search has finding a desired object as its goal.

6 Example

Having given an overview of the system's components, we now illustrate ZAROFF with an example drawn from a two-player game of hide and seek. To illustrate the conduct of a search, we will use an example environment with two buildings, one of which has two internal rooms separated by a door (Figure 8). Our example begins in the middle of a game, after the hider has hidden and the seeker has finished counting. The hider is in **state 2**, waiting to be seen. The seeker changes to **state 6** of its goal machine and begins to seek. The goal machine state specifies a "seek" goal to the planner as `goto(X)` with the added constraint that `type(X) = HUMAN`.

The ITPLANS planner considers the action `goto(X)` to be primitive but underspecified since

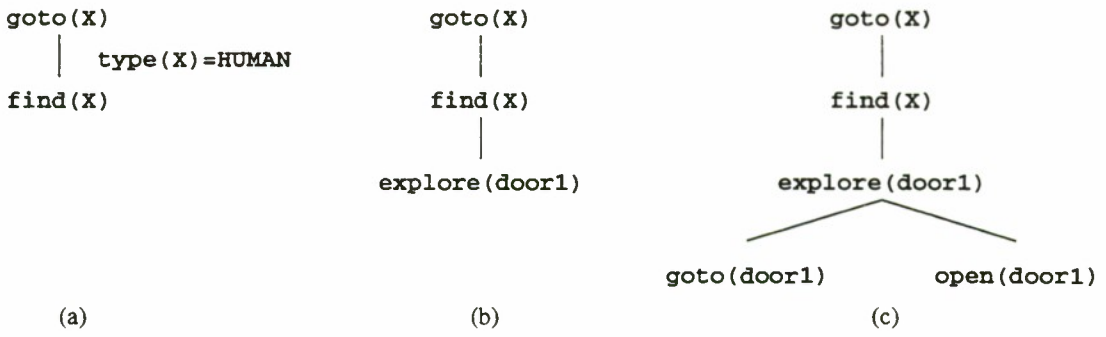


Figure 7: Evolution of the plan graph

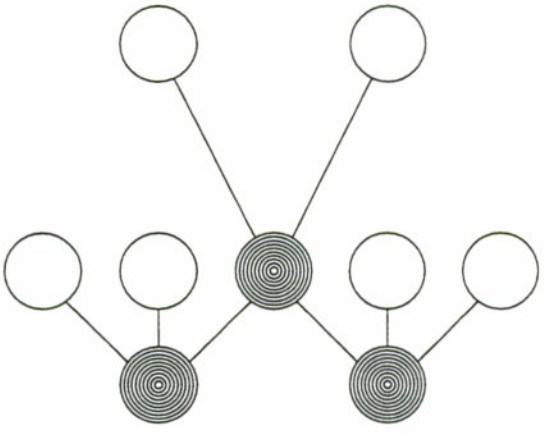


Figure 9: Initial map

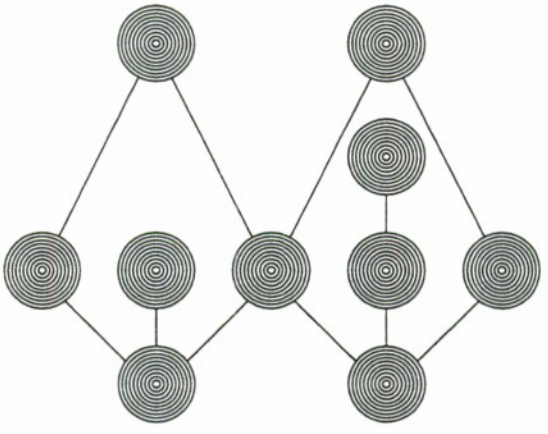


Figure 10: Final map

the variable **X** is not bound to a particular object of type **HUMAN**. In order to bind the variable, the search planner must be called to generate a plan for locating a **HUMAN**. To this end, ITPLANS adds to the plan a *find node* and calls the search planner to instantiate a search plan (Figure 7a).

The search planner reasons from this knowledge acquisition goal of locating a **HUMAN**, to the goal of exploring regions where a **HUMAN** might be. Satisfying this goal requires physically searching through possible regions.

ITPLANS asks the search planner to expand the find node. Each time a find node is expanded, the search planner first examines the *Jack* environment to determine if an object of the specified type is visible to the agent. If not, the search planner selects a region to explore next, generates a goal to explore that region, and adds it to the plan (Figure 7b). The initial map (Figure 9) of regions contains all the regions in the environment except the interior room of the building on the right. Regions that are completely visible are marked as having been explored; partially visible regions are marked for future exploration. The closest available unexplored region is the first room in the building on the right; it is recommended for exploration. This goal is then further expanded by ITPLANS to go to `door1` and open it (Figure 7c). Since all of the arguments in the first action are bound to specific objects, it can be carried out. Action Execution performs this action indirectly by binding an attraction sensor to the seeker. When the seeker arrives, `door1` is opened directly by Action Execution.

After `door1` is opened, ITPLANS uses the search planner to evaluate the progress of the search by examining the world for objects having the property **HUMAN**. If one is located, the search is considered successful. If not, the search planner selects a new region for exploration and the searching process repeats until there are no more regions to explore.

In this case, opening `door1` does not reveal a **HUMAN**, but does permit the agent to see another

region that is automatically added to the search planner's internal map. As this new region is the closest unexplored space, on the next iteration the planner will plan to explore it. Opening `door2` does not reveal the `HUMAN`, so the search proceeds to the next closest unexplored region, the right side of this building.

Here we see the advantage of maintaining a map. Immediately after opening `door2`, the agent is inside one building and decides to go to a non-neighboring region. Since this destination region is known (from having seen it previously), we could simply go there. This would result in the agent walking directly toward the destination until stopped by the wall. To avoid getting caught in this local minimum, the search planner uses its internal map (Figure 10) to plan a path to the next region.

The only known path there is to exit the current building through `door1`. The search planner returns this sequence of intentions to ITPLANS, which then invokes Action Execution to generate locomotion along this path. Eventually, after opening `door0`, the seeker finally sees a `HUMAN` and can go to it.

7 Conclusion

We have implemented a plan-based controller for a player in the game of hide and seek. The complete seeker is implemented, we are extending our architecture to implement the hider. Our agent dynamically reacts to changes in the environment, from avoiding collisions with obstacles and other players to exploiting changes in information about where the other players may be hiding. The implementation combines general purpose planning, special purpose reasoning about conducting a search, and reactive control of human behaviors.

ZAROFF is an effective system for animating humans carrying out tasks that require locomotion. Limiting the human agent's awareness of its environment by simulated perception increases the realism of the behavior generated.

8 Acknowledgements

We wish to thank our advisors Norm Badler and Bonnie Webber for their support of this work. Thanks also to them, Welton Becket, Jonathan Crabtree, Brett Douville, and Jeff Nimeroff for commenting on drafts of this paper.

9 References

Badler, Phillips, & Webber 1993 Badler, N.; Phillips, C.; and Webber, B. 1993. *Simulating Humans: Computer Graphics, Animation and Control.* Oxford University Press.

Becket & Badler 1993 Becket, W., and Badler, N. I. 1993. Integrated behavioral agent architecture. In *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*, 57–68.

Becket 1994 Becket, W. M. 1994. The Jack LISP API. Technical Report MS-CIS-94-01, University of Pennsylvania, Philadelphia, PA.

Geib 1992 Geib, C. 1992. Intentions in means-end planning. Technical Report MS-CIS-92-73, Department of Computer and Information Science, University of Pennsylvania.

Ko *et al.* 1994 Ko, H.; Reich, B. D.; Becket, W.; and Badler, N. I. 1994. Terrain navigation skills and reasoning. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representations.*

Ko 1994 Ko, H. 1994. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion.* Ph.D. Dissertation, University of Pennsylvania.

Moore, Geib, & Reich 1995 Moore, M. B.; Geib, C.; and Reich, B. D. 1995. Planning and terrain reasoning. In *AAAI Spring Symposium on Integrated Planning Applications.* (also University of Pennsylvania CIS department Technical Report MS-CIS-94-63/LINC LAB 280).

Reich *et al.* 1994 Reich, B. D.; Ko, H.; Becket, W.; and Badler, N. I. 1994. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, 996–1005. Geneva, Switzerland: IEEE Computer Society Press.

10 Biographies

Michael B. Moore is a Ph.D. candidate in Computer and Information Science at the University of

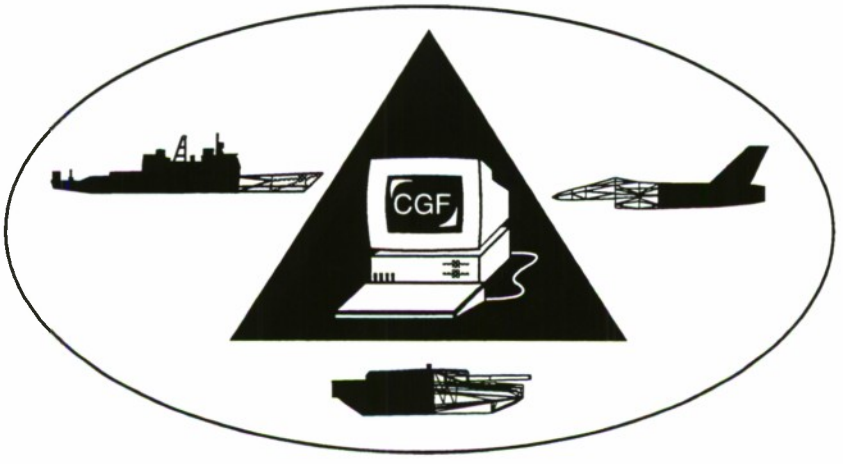
Pennsylvania. His current research in Artificial Intelligence is planning for information acquisition. He received his A.B. degree in Philosophy in 1986 from the University of Maryland and his M.S.E. degree in Computer and Information Science in 1990 from the University of Pennsylvania.

Christopher Geib is a post-doctoral research fellow at the University of British Columbia. His current research in Artificial Intelligence is planning. He received his Ph.D. degree in Computer and Information Science from the University of Pennsylvania in 1995.

Barry D. Reich is a Ph.D. candidate in Computer and Information Science at the University of Pennsylvania. His current research includes providing high-level behavioral control over the animation of human locomotion. He received his B.S. degree in Mathematics and Computer Science in 1989 from the University of Maryland and his M.S.E. degree in Computer and Information Science in 1991 from the University of Pennsylvania.

Session 7b: Terrain Modeling I

Hille, ANSER
Schaper, East Tennessee State University
Smith, Loral ADS


CGF

Abstracting Terrain Data Through Semantic Terrain Transformations

David Hille
ANSER
1215 Jefferson Davis Hwy
Arlington, VA 22202
hilled@anser.org

Michael R. Hieb Gheorghe Tecuci J. Mark Pullen
Department of Computer Science
George Mason University,
Fairfax, VA 22030
hieb@cs.gmu.edu tecuci@cs.gmu.edu mpullen@cs.gmu.edu

1. Abstract

Human commanders transform terrain from a map or from their personal observation into an abstract model used for reasoning. Automated commanders of CGF need to do a similar kind of transformation, since the data in a terrain database is too detailed. This paper describes *Semantic Terrain Transformations* developed during our work on the Captain project to transform data from a terrain database into concepts relevant to the mission given to the automated commander. Captain is an automated knowledge acquisition system that allows a subject matter expert to easily teach a command agent required behavior. Semantic Terrain Transformations first transform the digital data from the terrain database into an abstract geometric model (at the appropriate level of detail for the commander) and then translate the geometric model into symbolic concepts appropriate for reasoning. The concepts created are then used by an automated agent in performing its mission. We illustrate the methodology with detailed examples at both the battalion and company level. Captain has successfully learned rules for automated company commanders based on concepts generated by applying this methodology.

2. Introduction

We are currently developing a methodology and implementing a system, called Captain (Hille et al., 1994; Tecuci et al., 1994; Hieb et al., 1995) to construct command agents for Computer Generated Forces . This general approach offers advantages over the knowledge acquisition methods currently used for CGF behavior. Recent experiments with the Captain and ModSAF (Ceranowicz, 1994) systems led to the development of improved terrain representations more appropriate to command CGF than entity CGF. In order to derive these concepts from the CGF terrain database, we developed a method and set of transformations called Semantic Terrain Transformations. In this paper, we describe these transformations and illustrate them with two examples, from two different command levels.

The terrain database of a Distributed Interactive Simulation (Pullen, 1994) is typically in a form intended to satisfy several simulation goals. A terrain database represents a set of features such as soil, trees, roads, rivers, and buildings. It contains the elevations and relief of the terrain. The terrain database should be compact enough so that it does not occupy too much memory, allow for rapid access during a simulation, represent those features relevant to the real world events, and provide sufficient detail so that the world it represents seems realistic. It should simultaneously facilitate various processes such as observation, fields of fire, cover, concealment, and movement. No matter how well a terrain database satisfies all these goals, it is difficult to represent all the concepts an automated commander at any echelon might require in one database, without any transformation.

While general terrain reasoning methods have been described (Stanzione, 1993), most of the terrain reasoning methods in the literature are concerned with near-term movement (Cunningham, 1993, 1994; Smith, 1994), route-planning (Van Brackle, 1993) or cover and concealment (Longtin, 1994). Because of the size of typical CGF terrain databases and the need to process the data efficiently, solutions to these problems are generally algorithmic and focus on the entity level. However, an automated commander must be able to perform a wide range of missions adequately. This requires both a broad terrain reasoning capability and an ability to create more general and abstract concepts. Our approach is to build upon the work already done (e.g., use the techniques that calculate cover or avenues of approaches) and create a hierarchy of more abstract models that can be used by different echelons of command.

Human commanders must reason about the placement and movement of their forces. They subconsciously transform terrain they see on a map (as in Figure 1) or from their personal observation of a battlefield into an abstract model that they then use for reasoning as they make tactical decisions. Automated commanders need to do a similar kind of transformation, since the data in a terrain database may be too detailed for them to reason about it efficiently, and may not directly represent concepts pertinent to their decisions. When an agent (human or computer) makes decisions about terrain, the agent does not view terrain in terms of just elevations and relief. Instead, the agent reasons using abstract concepts such as hills (with ridges, forward slopes,

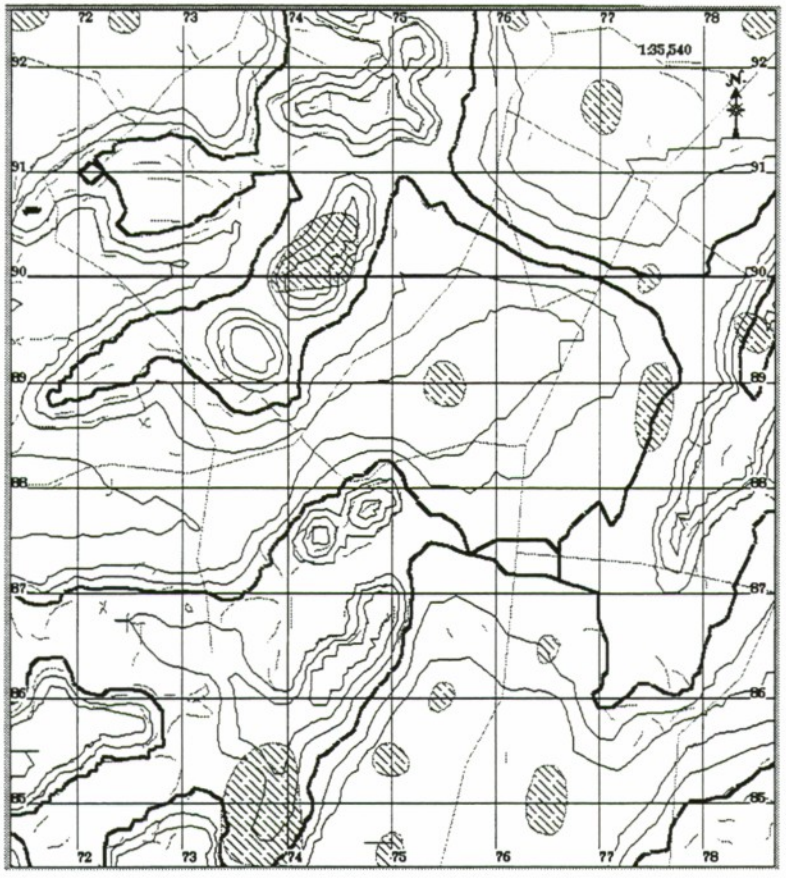


Figure 1: Terrain Map of Portion of Fort Knox, Kentucky

rear slopes, etc.), valleys, draws, wadis, spurs, saddles, depressions, cuts, fills, and cliffs.

In two extended examples, we illustrate our method by performing semantic terrain transformations on a terrain database in the context of an automated commander given a defensive mission. First, we transform detailed geometric data from a terrain database of Fort Knox, Kentucky into a geometric model at an appropriate level of abstraction, given the level of the commander (e.g., company, battalion, etc.). In a second step we form relevant concepts from the abstract geometric model. The result of our semantic terrain transformation process is a semantic network containing information an automated agent needs for efficient planning and learning. See (Tecuci, 1992) for an explanation of semantic networks.

The first example is at the battalion level. Terrain concepts are created to enable the automated battalion commander to form company sector boundaries in a sector defense. The second example uses the transformations to support an automated company commander as it places the tank and mechanized infantry platoons of a U.S. Army company to defend a company sector against an expected attack by an opposing force, given sector boundaries and an avenue of approach, from the automated battalion commander. A more detailed description of the second example is given in (Hieb et al., 1995), which concentrates on learning a placement rule after the Semantic Terrain Transformations have been performed.

The rest of this paper is organized as follows. Section 3 presents the general methodology of the Captain approach. Section 4 describes the Semantic Terrain Transformation process. Section 5 presents an extended example at the company level. Finally, Section 6 concludes the paper with a discussion of our terrain reasoning approach.

3. Captain Methodology

Captain is a methodology for building adaptive command agents for CGF. The Captain implementation includes an apprenticeship learning system, which combines machine learning and knowledge acquisition methods (Tecuci 1988; Tecuci 1992). Captain creates adaptive command agents in an integrated framework that facilitates both 1) building agents through knowledge elicitation and interactive apprenticeship learning from subject matter experts, and 2) making these agents adapt and improve during their normal use through autonomous learning.

In the Captain methodology (Tecuci et al., 1994), we define three phases in the creation of an agent (see Figure 2).

In the first phase, Knowledge Elicitation, the subject matter expert (SME) works with a knowledge engineer to define an initial knowledge base (KB) which will contain relevant background concepts and relationships. This KB is expected to be incomplete and partially incorrect at this point. Semantic Terrain Transformations are an essential part of building this initial knowledge base, to develop the terrain concepts and transformation rules required for a type of mission.

In the second phase, Apprenticeship Learning, the Command Agent will interactively learn from the SME by employing apprenticeship learning techniques (Hieb et al., 1995). This consists of showing the SME instances of a type of mission (e.g., defend in sector for a company command agent), and learning how to produce the orders to accomplish the mission. The result of this process is a set of rules that can be used to generate orders for similar types

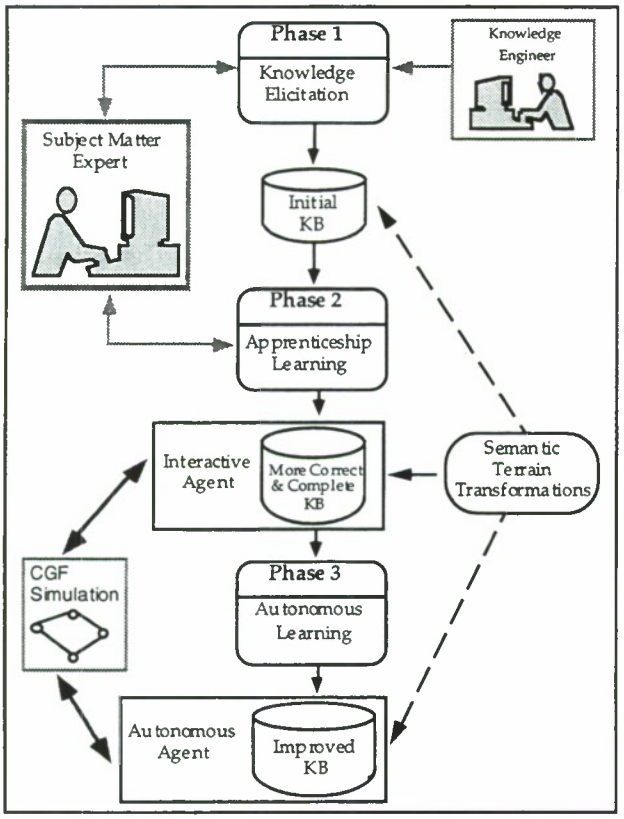


Figure 2: The Main Stages of Building a Captain Agent

of missions. These rules can be translated into a more efficient form for use during problem solving. Semantic Terrain Transformations will be utilized repeatedly, to process the terrain considered for each specific mission.

When the Captain command agent has been trained with examples of the typical ModSAF problems it should be able to solve, it enters a third phase, Autonomous Learning, where it is used in simulations without the assistance of the SME. However, the agent will continue to learn from its own experiences by employing the same multistrategy learning techniques it used when learning from an SME.

Semantic terrain transformations are an integral part of both building the initial knowledge representation, and in the learning process used by Captain in phases 2 and 3.

4. Semantic Terrain Transformations

Captain agents use a hybrid knowledge representation integrating semantic networks and rules. Semantic networks represent the information from a terrain database at a conceptual level, as well as generic and specific knowledge about weapon systems and forces. Rules are used to represent the behavior and decision making of the agent as it generates orders for accomplishing missions.

The detailed *geometric representation* is the source of the data to be transformed (e.g., ctdb database). The transformation process is based on background knowledge, a decision context, and a set of relevance criteria. The *background knowledge* defines the concepts pertinent to the transformation process. The *decision context* is the set of circumstances at the time a transformation takes place. This context is based on the mission the automated agent has been assigned and the current situation of the agent. A variety of abstract models may be generated from a given terrain database, depending on context. The *relevance criteria* identify which features and which levels of detail need to be reflected in the target model. A critical component of the semantic terrain transformation process is the categorization of each component of the more detailed geometric model as important or unimportant, based on the context of the decision. The relevance criteria provide metrics for identifying what features may be deleted or simplified. The *abstract geometric representation* reflects the set of concepts to be used in the decision process, where each concept is represented at a level of detail appropriate for satisfying decision goals.

Transformations are accomplished by the successive application of operators. These operators are four kinds of elementary knowledge transformations: *abstraction, generalization, aggregation, and simplification.*

The abstraction process involves selectively removing "unimportant" features from a model. Features are unimportant if they have little impact on the quality of decisions made in a given context, when using the resulting model (i.e., the relevance criteria considers them unimportant). For example, individual trees may be removed when transforming a detailed terrain database into a representation appropriate for use by a company commander, since the location of individual trees is usually not relevant to the decisions of the company commander.

The generalization process replaces the representation of individual features with the representation of classes of features. For example, instead of representing the unique effect of each piece of road on the movement of a unit, road segments may be categorized and an effect associated with each category of segment.

Aggregation involves summing up detailed data and representing it in aggregated form. For example, a line of hills is composed of different parts: a ridge crest running along the tops of the hills, the front slope of the line of hills (relative to the observer), the rear slope, and side slopes. The ridge crest, in turn, is

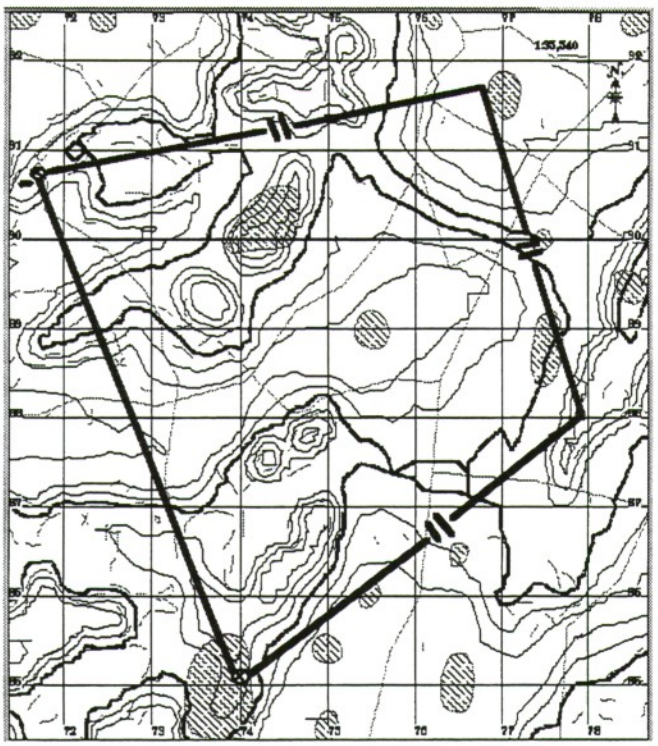


Figure 3: Situation at Start of 3rd Battalion Mission

composed of the high areas of the line of hills, spurs jutting out from the sides of the ridge, and saddles (dips in the ridge crest). The aggregation of indivdual features produces composite features or superfeatures (Stanzione, 1994).

Simplification is similar to abstraction in that it eliminates some details, but it also may include some distortion of unimportant details, in the process of producing a more compact knowledge representation. For example, the complex shapes of contour lines may be replaced with simpler shapes.

In the examples in this paper, we identified those features that could be abstracted out of the model and used simplification to reduce the shapes of the regions to simple geometric objects, including line segments and rectangles. The resulting geometric model contains only physical features and does not include individual elevations or trees, for instance.

4.1. Transformation Phases

Semantic terrain transformations constitute the process of converting terrain information from a terrain database into a semantic network for use by an automated command agent and consists of two phases.

In the first phase, the geometric representation from a terrain database is iteratively transformed into one or more increasingly abstract geometric models. These models are more compact and contain higher level concepts needed for reasoning efficiently.

In the second phase, the geometric model at the appropriate level of abstraction is translated into a textual representation in the form of a semantic network. The semantic network describes the geometric information as a set of concepts and the relationships among the concepts. The textual representation in the semantic network may then be used by an automated agent in making decisions.

4.2. Generation of the Abstract Geometric Model

In this phase, we transform detailed geometric data from a CGF terrain database of Fort Knox, Kentucky into a geometric model at an appropriate level of abstraction, given the decision-making context. The situation confronting an automated commander (of the 3rd Battalion) is represented in Figure 3. In this figure, changes in elevation in intervals of 10 meters are depicted by contour lines.

There are three separate problems the automated battalion commander must solve using the semantic network. First, the agent must choose the appropriate form of defense. Its options include company sector defense, company battle position defense, or company strong point defense. Second, the agent must establish company areas of responsibility. Finally, given its choice of defense, the agent must establish decide which particular company will occupy which area of responsibility.

In choosing the appropriate form of defense, the automated battalion commander uses rules that require a company sector defense if it cannot concentrate fires (as when there are multiple avenues of approach such as in our example). There are three avenues of approach leading into the 3rd battalion's area of responsibility. The battalion commander accordingly chooses a sector defense with one company sector per avenue of approach and one of the two tank companies in reserve to the rear of the company areas. During a battle, the reserve company may be used in local counterattacks or may be moved to bolster the defense of a company being attacked.

In this situation, an automated battalion commander has decided to use a sector defense. Defense of a sector is the most common defense mission for a battalion in contemporary combat missions. The sector is an area designated by boundaries that form an area of responsibility. Sectors are generally deeper than they are wide to permit the defending unit to fight the battle in depth. In the current situation, the enemy forces are expected to approach from the Southwest of the battalion's area of responsibility. The automated battalion commander must then perform a set of semantic terrain transformations to determine the company sector boundaries within it's area of responsibility.

The process of transforming a detailed terrain model into an abstract terrain model consists of four steps:

Step 1: Identify decision context.
Step 2: Identify relevance criteria.
Step 3: Establish new concepts to be represented in the target terrain model and generalization hierarchies for terrain regions.
Step 4: Apply transformation operators to the current terrain model to produce the target terrain model.

Step 1 involves making explicit the parameters and types of operations affecting the transformation process. These influence the application of the relevance criteria. In our example, we identify the decision context as the placement of the companies of a battalion. The context includes the primary decision factors: mission, enemy, troops, terrain (and weather), and time available (METT-T). In this situation, the mission of the battalion commander is to defend a battalion sector against an expected attack by an opposing force expected to consist of two or more tank or motorized infantry battalions. The battalion sector boundaries have been identified and are reflected in Figure 3. The troops available to the 3rd Battalion commander are two tank and two mechanized infantry companies of a battalion task force.

Step 2 determines the relevance criteria that are used to determine the forms of concepts to be included in the target terrain model. The resulting relevance criteria determine the set of transformation operators that are needed to perform the transformations and their sequence. The relevance criteria may be expressed in a set of rules. These rules may be learned by command agents using the Captain methodology as explained in (Hieb et al., 1995). An example of a rule defining a relevance criteria might be: "If a terrain feature cannot significantly affect mobility of subordinate units and the terrain feature cannot significantly affect vision of subordinate units and the terrain feature cannot significantly affect concealment of subordinate units then the terrain feature is irrelevant." If this rule is applied in a situation where the terrain feature is an individual tree and subordinate units are companies, since an individual tree affects neither the mobility, nor the vision, nor the concealment of a company as a whole to a significant degree, any individual tree would be considered irrelevant and excluded from an abstract terrain model at that level. On the other hand, if the subordinate unit is an individual entity, a tree would be relevant since it may offer concealment to the entity and may affect its field of view.

Granularity refers to the level of detail of representation for the objects in the target terrain model. There is a trade-off between the higher quality of decisions that accompany fine granularity and the greater speed of decision-making that accompanies coarser granularity. For reasoning about the placement of companies in the battalion we chose the level as about 1/1000th the area of the model. The area the model represents is about 7 kilometers by 8 kilometers, or about 56 square kilometers. Dividing this by 1000, a size of about 1/16 of a square kilometer is established as the model granularity. That is, shapes are simplified into the nearest 1/16th square kilometer (1/4th kilometer by 1/4th kilometer).

Step 3 requires establishing generalization hierarchies for terrain regions. A terrain region may be classified as a physical region or an organizational region. Physical regions are classified according to the physical properties of the terrain that affect the accomplishment of goals. Organizational regions correspond to regions established in map overlays. In past military operations, an overlay was a transparent medium on which information was plotted on top of a map, photograph, or other graphic. In operations supported by map automation, the overlay information is plotted on top of information from the terrain database. It reflects unit boundaries, routes, areas of responsibility, engagement areas, etc.

We identify three physical and three organizational subclasses of regions: 1) the physical relief regions in which terrain is classified as being in hills or relatively flat areas; 2) the physical cover regions in which terrain is aggregated into regions based on the presence of natural or man-made forms of cover and concealment; 3) the physical mobility regions in which terrain is aggregated into regions based on the presence of natural or man-made obstacles or features such as roads that enhance mobility; 4) the organizational avenues of approach, each of which consists of a mobility corridor and an engagement area; 5) the organizational regions that define the area of responsibility for the command agent; and 6) the organizational regions defining boundaries of subordinate units. Within each region subclass, each region is discrete; there is no overlap.

Figure 4 shows the abstract model resulting from Phase 1 of the transformations. It displays four of the six primary regions in one diagram. The physical mobility regions were omitted since they correspond closely to the avenues of approach displayed. Also, organizational regions defining boundaries of subordinate units were omitted since these boundaries are not determined until after phase 2 of the semantic terrain transformations has been completed and the battalion commander has decided the sector boundaries of its companies.

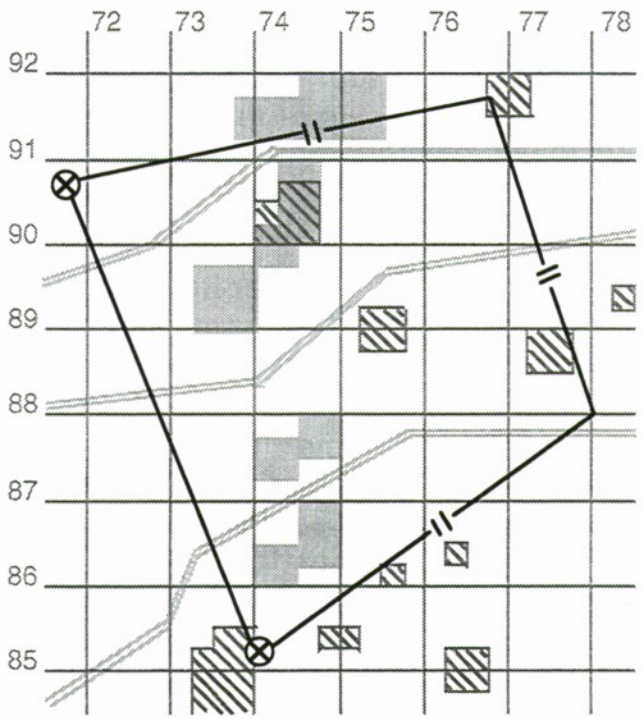


Figure 4: Abstract Model.

Hills and various other terrain features such as valleys, ridges, and depressions are concepts that may not be explicitly represented as such in a terrain database. They may be determined by examining contour lines. For the purpose of terrain transformations, we defined a hill more precisely as an area where there are at least three concentric rings of contour lines (total elevation change of 30 meters) enclosing an area less that one square kilometer whose average slope is greater than 12° .

Step 4 maps from the source terrain model to the target terrain model. The process uses operators to accomplish a homomorphic mapping from the features in the source terrain model to the features in the target terrain model.

Figure 4 is an abstract model of parts of the terrain database for use in making decisions by a battalion commander. Each square in the grid represents one kilometer. The light gray rectangles are hills, dark gray rectangles with slanted lines are tree canopies, lines with double hash marks denote the battalion area of responsibility, and x's inside circles denote the forward edge of battle area (FEBA).

The avenues of approach are identified using existing terrain analysis programs.

At this point in the example, the abstract geometric terrain model contains information at a level of abstraction appropriate for making decisions. For an automated agent to reason about the information, it needs to be translated into objects and relationships in a semantic network. This is done in the second phase of the semantic terrain transformation process.

4.3. Concept Identification

In the second phase of the transformation process we transform information from the abstract geometric model into concepts and relationships among concepts in a semantic network. This process requires identifying appropriate types of concepts and relationships and the application of further knowledge transformations.

This process consists of four steps:

Step 1: Assign a name to each separate object in the geometric terrain database.

Step 2: Establish the set of relationships to be used to represent relationships among objects in the terrain model.

Step 3: Establish relevance criteria for choosing which ordered pairs of objects named in Step 1 will be associated with each kind of relationship described in Step 2.

Step 4: Create the semantic network for the command agent.

Step 1 assigns names to objects. Some of the names assigned are shown in Figure 5.

In Step 2, the set of relationships among objects is identified and defined based on the decision context. The relative location of opposing forces to the friendly unit is used to orient objects in terms of "LEFT," "RIGHT," "IN-FRONT-OF," and "BEHIND." Relationships such as "NEXT-TO," "NEAR," and "FAR" are defined based on number generalization, which maps distances between objects to general concepts.

In Step 3, the relevance criteria are established based on mission and level of the unit. They may be expressed as a set of rules that may be learned by the agent (Hieb et al., 1995).

In our example, the concept of "area-of-responsibility" had already been defined in the in the agent's knowledge base, as well as a rule for "WITHIN," which determined whether an arbitrary object was contained in the battalion area of responsibility. One relevance criterion holds that only terrain objects (physical regions) contained within the area of responsibility of the battalion may be related to other objects in the semantic network. Other relevance criteria pertain to distance.

In Step 4, the relevant information, representing interrelation-ships among objects from the geometric

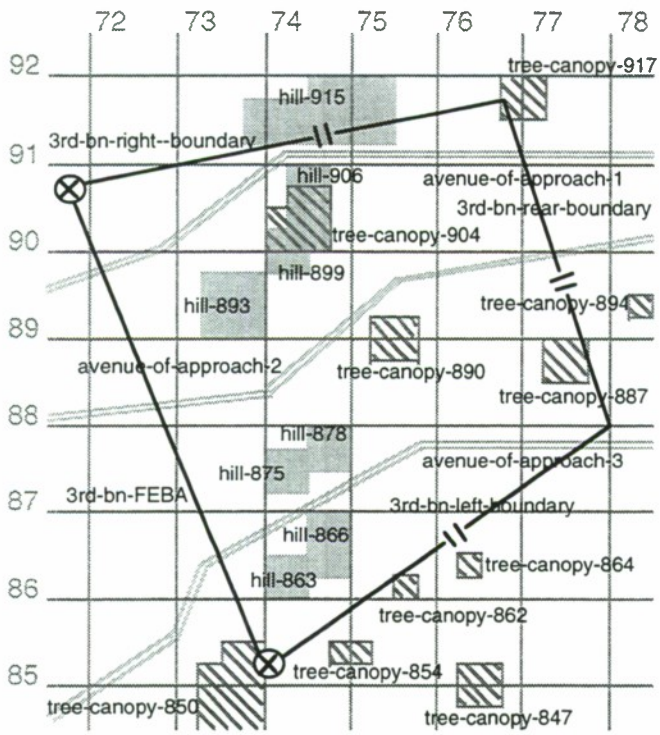


Figure 5: Selected Object Names

terrain model, is entered into the semantic network. A small part of the resulting semantic network is shown in Figure 6. The gray unlabeled arrows in the diagram indicate instance-of relationships.

The semantic network in the agent's knowledge base represents concepts that are used by the agent during learning. In the case of the 3rd Battalion commander given the defensive mission, Captain will work with the SME to learn a rule that establishes company sector defense boundaries.

In positioning company sector boundaries, the SME may teach the automated battalion commander to orient the boundaries on the avenues of approach to provide dominating terrain for each company. The result of this learning process would be a rule that establishes company sector defense boundaries as in Figure 7.

5. Company Terrain Transformations

In the previous section, the automated battalion commander used Semantic Terrain Transformations to establish company sector boundaries for the four companies under its command. The automated battalion commander then would issue orders to automated company commanders to establish defensive positions in the sectors assigned.

To carry out the order from the battalion commander, each company commander must reason about the area of terrain for which it is responsible. The company commander uses a set of Semantic Terrain Transformations to put the terrain information into a useful form. The terrain model used by the battalion commander does not reflect all the concepts a company commander needs to consider nor are the concepts in the battalion's model in sufficient detail. In this section we will describe how Semantic Terrain Transformations may be done by a company commander who has received a defend sector mission from its battalion commander.

The area of responsibility of the company

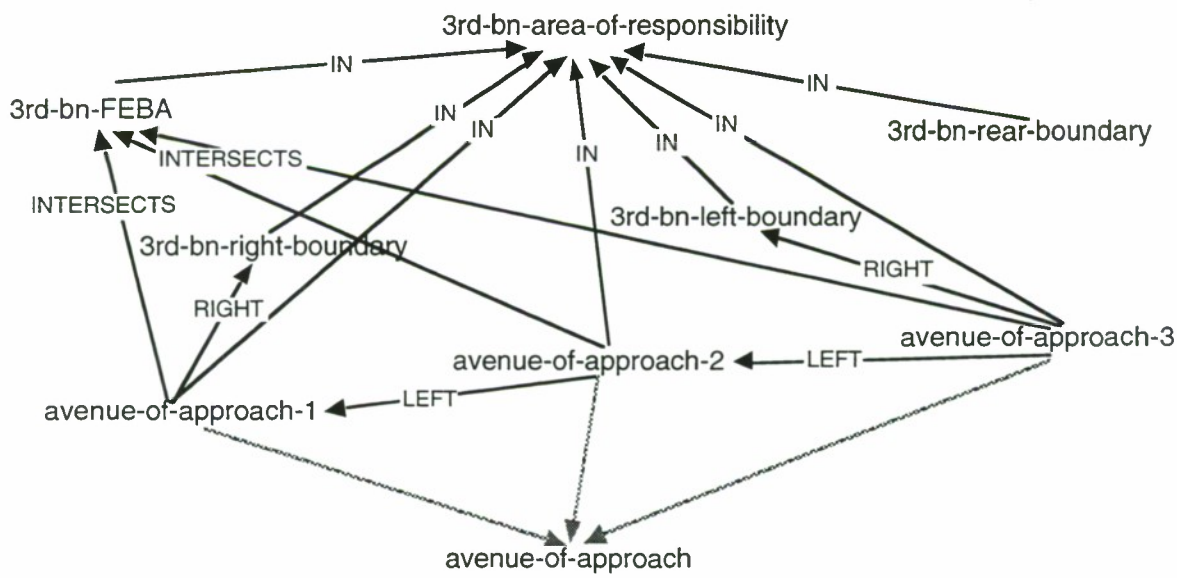


Figure 6: Portion of Battalion Semantic Network

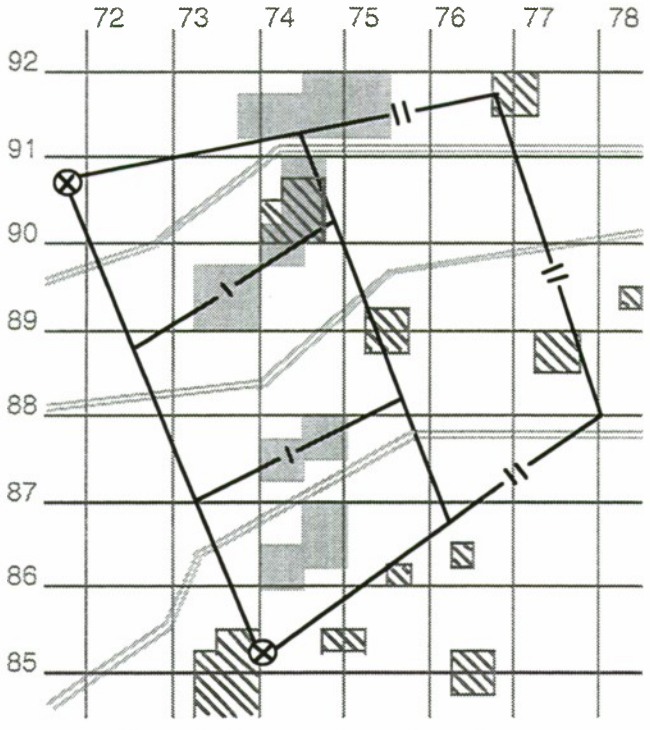


Figure 7: Company Sector Boundaries

commander is shown in Figure 8a. It is the Southern area of responsiblity shown in Figure 7. To change the data from the terrain database into a form that can be effectively used by the company commander, the commander employs Semantic Terrain Transformations using the same algorithm discussed in the previous section but applied in a much different context. The company command agent applies the four steps of phase 1 in generating an abstract geometric model.

In Step 1 the decision context is the mission of the company commander to defend a company sector against an expected attack by an opposing force consisting of one or two tank or motorized infantry battalions. The troops available to the commander are two tank and one mechanized infantry platoons.

In Step 2 of the algorithm, the commander determines what must be included in the target terrain model. The existing features considered relevant for the current decisions are avenues of approaches, engagement areas, and hills.

In Step 3, the agent generates needed concepts such as the concepts of hills and various types of hill features. A hill from the perspective of a company commander has the same basic definition as that used by the battalion commander. However, a hill is represented differently for the company commander than for the battalion commander. The main difference in the representation of hills between the two is that the company commander needs a finer granularity of the representation and must also represent hill parts such as front slope (relative to the avenue of approach), rear slope, and crest.

A hill is represented in the abstract terrain model of a company commander as a circle. Long hills are represented as a series of connected circles, as in Figure 8b. To reflect the locations of front and rear slopes and the crest of the hill, each hill is twice bisected to form four quadrants. The first two quadrants face the avenue of approach while the last two quadrants face away from it. Each hill quadrant has a special set of features relevant to the company commander's decisions and is thus considered a separate object in the agent's knowledge base.

The avenue of approach is obtained from the automated battalion commander, while the engagement area will be calculated using existing terrain analysis programs.

Various other concepts were generated at this point such as the visibility of one region to another region, using line of sight algorithms commonly implemented in CGF systems. Distances from one region to another were defined as close, near, far, or remote based on the weapons systems of the type of unit and other factors.

In Step 4 of the first phase, the agent applies transformation operators to generate the terrain model shown in Figure 8c.

In the second phase of the transformation process the map is transformed into a symbolic form expressing concepts and relationships in a semantic network. This process included labeling objects, identifying relevant relationships and entering the concepts into the semantic network as shown in Figure 9. Several of the representation units in the knowledge base are shown in Figure 8d.

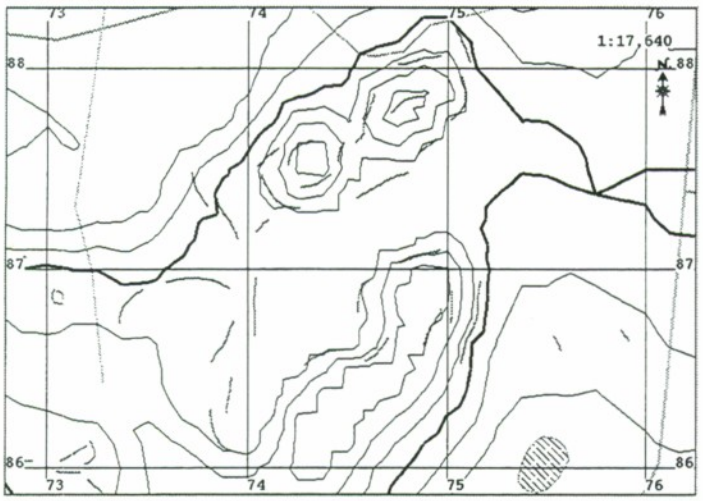
The representation units have the notation:

```
(concept-i concept-k   (FEATURE-1 value-1)
                                  . . .
                       (FEATURE-n value-n))
```

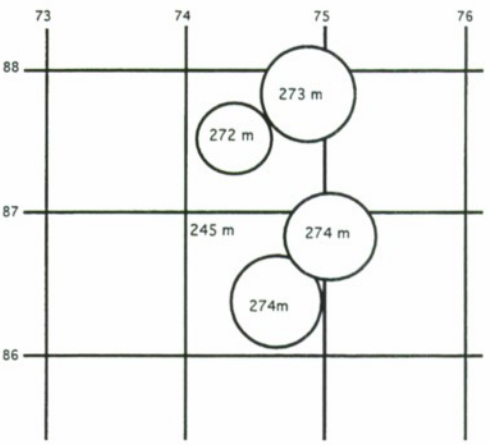
This expression defines "concept-k" as being a subclass of "concept-i" (from which it inherits features) with additional features. The value of a feature may be a constant or another concept.

Based on the concepts in the semantic network resulting from the Semantic Terrain Transformations, the automated company commander is able to make effective decisions. Our method of teaching such decision rules to automated command agents is described in (Hieb et al., 1995).

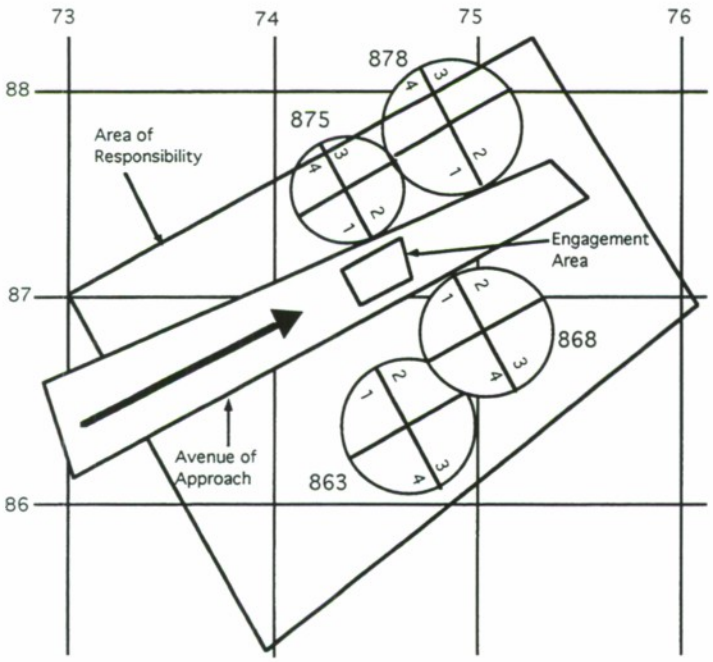
8a



8b



8c



8d

```
(hill  hill-863
    (orientation "right")
    (across hill875)
    (across hill878)
    (front hill868)
    (size 3))

(hill-sector  hill-sector-863-1
    (quadrant 1)
    (visible mobility-corridor-d)
    (visible engagement-area-d)
    (in company-d-area-of-responsibility)
    (part-of hill863)
    (distance-to-engagement-area "close"))

(hill-sector hill-sector-863-2
    (quadrent 2)
    (visible mobility-corridor-d)
    (visible engagement-area-d)
    (in company-d-area-of-responsibility)
    (part-of hill863)
    (distance-to-engagement-area "close"))
...
```

Figure 8: Company Semantic Terrain Transformations

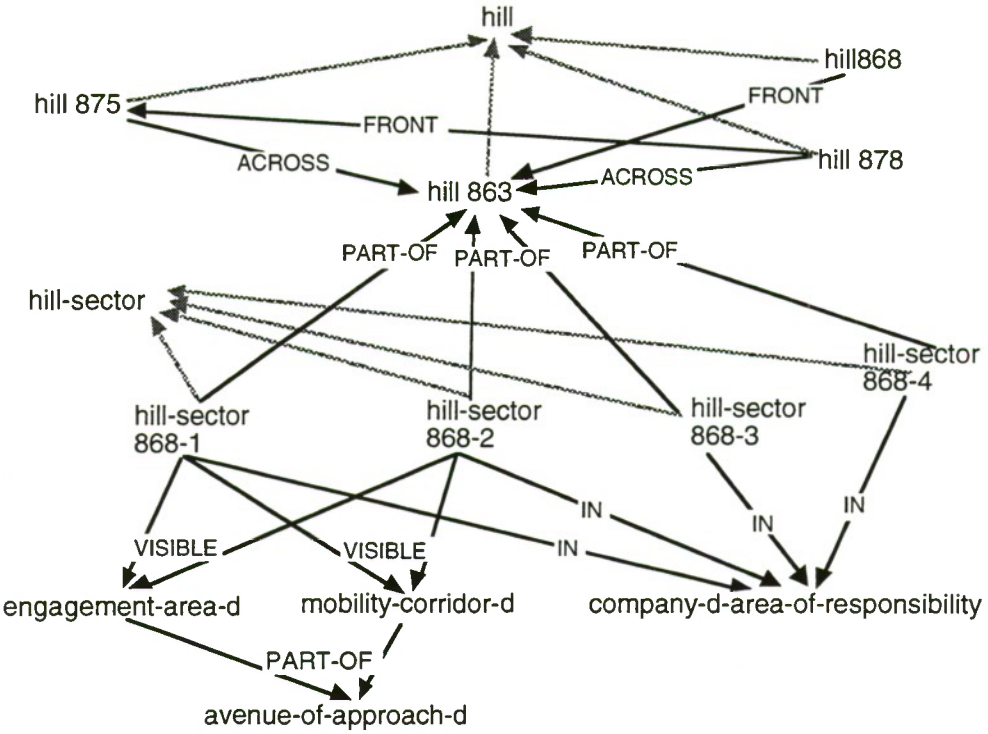


Figure 9: Portion of Company Semantic Network

6. Conclusions

In this paper we have presented a methodology for transforming a terrain database into a semantic network which is used by automated command agents to perform terrain reasoning. We have applied this methodology to generate the semantic network representation of the terrain necessary for teaching ModSAF company commanders defensive missions, as described in (Hieb et al., 1995). We are currently refining the methodology of semantic terrain transformations and automating this process as part of the development of the overall Captain methodology.

7. Acknowledgments

This research was conducted in the Computer Science Department and the Center for Excellence in Command, Control, Communications & Intelligence at George Mason University. Work on ModSAF applications was sponsored in part by the Defense Modeling and Simulation Office under contract DCA100-91-C-0033.

8. References

Ceranowicz A., (1994). ModSAF Capabilities, *Proceedings. of 4th Conference on Computer Generated Forces and Behavior Representation,* Orlando, Florida.

Cunningham, C.T. (1993). Control of Movement in an Arbitrary Polygonal Terrain. *Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Cunningham, C.T. (1994). Development of Intelligent Simulations at LLNL. *Proceedings of the 4th Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Hieb, M.R., Hille D. and Tecuci, G. 1993. Designing a Computer Opponent for War Games: Integrating Planning, Learning and Knowledge Acquisition in WARGLES. In *Proceedings of the 1993 AAAI Fall Symposium on Games: Learning and Planning*, AAAI Press Technical Report FS-93-02, Menlo Park, CA.

Hieb, M.R., Tecuci, G., Pullen J.M., Ceranowicz A., & Hille D.(1995). A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces. *Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Hille D., Hieb, M.R. & Tecuci, G. (1994). Captain: Building Agents that Plan and Learn. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida.

Longtin, M.J. (1994). Cover and Concealment in ModSAF. *Proceedings of the 4rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Pullen, J.M. (1994). Networking for Distributed Virtual Simulation. In B. Plattner & J. Kiers Eds, *Proceedings. of INET'94/JENC5.* Internet Society (isoc@isoc.org).

Smith, J. (1994). Near-term Movement Control in ModSAF. *Proceedings of the 4rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Stanzione, T., Smith, J.E., Brock, D.L., Mar, J.M.F. & Calder, R.B. (1993). Terrain Reasoning in the ODIN Semi-Automated Forces System. *Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Stanzione, T. (1994). Suitability of the Standard Simulator Database Interchange Format for Representation of Terrain for Computer Generated Forces. *Proceedings of the 4rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

Tecuci G. (1992). "Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base," *IEEE Transactions of SMC,* 22(6).

Tecuci, G., Hieb M.R., Hille D. & Pullen J.M. (1994). Building Adaptive Autonomous Agents for Adversarial Domains, *Proceedings of the AAAI 94 Fall Symposium – Planning and Learning: On To Real Applications.* November

Tecuci G., Kedar S. & Kodratoff Y. (Eds), (1994). *Knowledge Acquisition* Special Issue on the Integration of Machine Learning and Knowledge Acquisition, 6(2).

Van Brackle, D.R., Petty, M.D., Gouge, C.D. & Hull, R.D. (1993). Terrain Reasoning for Reconnaissance Planning in Polygonal Terrain. *Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation.*, Orlando, Florida.

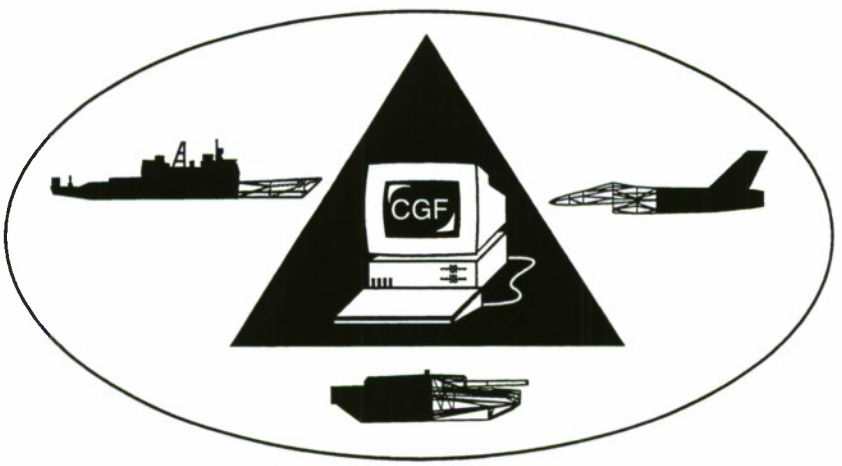
9. Authors' Biographies

David Hille is a computer scientist at ANSER, a public service research institute. He received a Masters of Science in Computer Science at Syracuse University and is currently a Doctor of Science candidate in the Information Technology PhD program at George Mason University. He has designed military simulations published commercially by Strategic Simulations, Inc. and helped to develop the wargame methodology for technology base seminar wargames. He has been working on the Captain Learning and Planning System project, a system that performs as an automated agent in simulations.

Michael Hieb is a PhD candidate in Information Technology at George Mason University in Virginia. He is currently a researcher at the Computer Science Department of George Mason University working on automated knowledge acquisition of behavior in complex domains, such as ModSAF. He is also researching interaction modes for knowledge acquisition interfaces. He has published papers in the areas of knowledge acquisition, multistrategy learning, and plausible reasoning. He has served as a consultant in AI for CSC and implemented a distributed problem solving testbed at the Goddard Space Flight Center for IntelliTek, Inc.

Dr. Gheorghe Tecuci is Associate Professor of Computer Science at George Mason University. He has published over 70 scientific papers, mostly in the area of artificial intelligence. Gheorghe Tecuci is a member of the Romanian Academy and is known for his pioneering work on multistrategy machine learning and its integration with knowledge acquisition. He developed Disciple, which is one of the first multistrategy learning systems, and co-edited the first books on multistrategy learning and on the integration of machine learning and knowledge acquisition. He was the program chairman of the first workshops in these areas (MSL-91, MSL-93, IJCAI-93: ML & KA).

Dr. J. Mark Pullen is Associate Professor of Computer Science at George Mason University. He also has an appointment with the Center for Excellence in Command, Control, Communications and Intelligence. Dr. Mark Pullen was employed by the Defense Advanced Research Projects Agency (DARPA) from 1986 to 1992, where he was Program Manager for Advanced Computing, Networking and Distributed Simulation, and Deputy Director of the Tactical Technology Office and the Information Science and Technology Office. His research interests include distributed and parallel computing systems and their applications to educational and military simulations.


CGF

Terrain Reasoning by Intelligent Player

Ashok Pandari and Gregory A. Schaper
Department of Computer Science
East Tennessee State University
Johnson City, TN 37614-0002

1. Abstract

Intelligent Player (IP) is a computer generated fighting helicopter that uses a game tree for determining combat maneuver decisions. It has been demonstrated that IP is a viable real-time simulation entity that is capable of formulating plans in real time. Planning gives IP the capability of selecting a maneuver to implement that might result in a temporary tactical disadvantage while achieving a superior tactical advantage in the future. Previous implementations of IP have focused on investigating the feasibility of computing a plan in real time. In this paper we expand previous implementations by incorporating terrain into the planning computation of IP. Experiments show that even when IP performs terrain avoidance it is still capable of formulating a plan in real time.

2. Introduction

Intelligent Player (IP) is a computer generated fighting helicopter used in the simulation of air-to-air combat. IP computes combat maneuvers in real time using a game tree [5] lookahead computation. IP's main utility is the computation of high fidelity maneuvers during close quarters, one-on-one, air combat. In a simulation environment with many entities, a workstation running IP code can be used for controlling helicopter entities involved in air-to-air combat. Other automated simulation techniques can be used for other portions of combat mission simulation.

Katz first defined IP in [2] [3] [4] and later provide an improved version of IP based upon differential game theory [1]. This improved version of IP has the significance of separating the issues of air combat, such as vehicle control, from those involved in the lookahead computation. Schaper, Pandari and Singh [9] implemented this improved version of IP with specific interest in determining the maximum amount of lookahead that could be computed in real time on current PC and workstation platforms. Their investigation revealed IP to be a viable real-time simulation entity. Specifically, it was shown that IP is a tenacious and aggressive opponent with the capability of formulating plans during lookahead computation. A plan is the ability to select a maneuver to carry out which will result in a temporary tactical disadvantage but will ultimately result in a tactical advantage.

In this paper we expand the current IP implementation to include terrain avoidance. In previous IP implementations, IP simulations were carried out in a three-dimensional void. We investigate the behavior and lookahead performance of IP when terrain avoidance is performed. Results of experiments based upon this implementation show that IP continues to behave in an aggressive and tenacious fashion. We also show execution timing results on RS6000 workstation which show that even when terrain avoidance is performed, IP is still capable of planning in real time.

Section 3 discusses terrain modeling techniques used in our IP implementation. Section 4 details specific modifications to existing IP implementation required to perform terrain avoidance. Section 5 presents measurements of IP's behavior and performance. Section 6 contains concluding remarks.

3. Terrain Modeling

Several techniques for modeling terrain have been used in combat simulation. Polygon Terrain Representation [8] and Digital Terrain Model [6] are two common techniques. When considering the use of a terrain modeling technique there are several key attributes that must be addressed. These attributes include the amount of memory used to represent the terrain, the fidelity of the representation, and the computation time required to look up terrain features during simulation exercises. One major limiting factor in the IP implementation described in this paper is the amount of available

memory on the RS6000 workstations. This forced a choice of terrain modeling technique that was memory efficient in representing terrain with reasonable fidelity and modest look up speeds. Polygon Terrain Representation and Digital Terrain Model require large memory for terrain representation. For this reason we choose to use quadtrees.

3.1 Quadtrees

A *quadtrees* is a data structure that makes efficient use of storage in representing cartographic databases that involve large amounts of data. Quadtrees are a special type of tree data structure in which each node, except the leaf nodes, have an outdegree of four.

At level one of a quadtree there is one node, called the root, of the quadtree. This node represents an extent of square terrain. This square extent is subdivided into four equal sized subquadrants, namely, northeast (NE), northwest (NW), southeast (SE) , southwest (SW). Each of these subquadrants are represented by a child node of the root and holds relevant information about its subquadrant. This decomposition continues recursively until all points in the subquadrant fall within a single plane. A least square error computation is used to determine if all points of the quadrant can be represented by a single plane.

Each node of the quadtree stores the coordinates of the lower left corner and the upper right corner of the quadrant it represents. Each node also contains pointers to each of its four child nodes and a pointer to its parent node. During construction of the quadtree, each node requires access to all terrain points which fall within the associated quadrant. This access is provided by a pointer to a list containing all points which belong to the quadrant.

Each node of the quadtree is processed to determine if all points of the subquadrant form a plane. If all points do form a plane then this node is a leaf. Otherwise, the node's quadrant will be recursively subdivided.

A plane is represented by an equation of the form

$$z = Ax + By + C. \tag{1}$$

The A, B, and C coefficients of this equation are stored in the associated node. If all data points have the same z value then A and B are assigned 0. If the data points do not have the same z values then a least square error computation is used to determine if the points in this quadrant lie along a single plane.

3.2 Least Square Error Computation

To determine if a quadtree node is a leaf node, it must be determined if the coordinates in the associated quadrant can be represented by a single plane. First, a plane equation of form specified in (1) is determined by using partial differential equations. The A, B, and C coefficients of the plane equation are computed by taking partial derivatives of the least square error equation

$$E^2 = \frac{\sum_{i=1}^{n}(z_i - (Ax_i + By_i + C))^2}{n}. \tag{2}$$

Coefficient A is computed by taking the partial derivative of E^2 with respect to A and equating the result to 0. Similarly, the coefficient B is computed by taking the partial derivative of E^2 with respect to B and equating the result to 0. After A and B are known then C can be easily determined.

Once the best-fit plane equation has been found, the least square error, E^2 is computed. E^2 is compared to a predefined tolerance value. If E^2 is less than the tolerance then the plane is created using A, B, and C as it's coefficients. Otherwise the quadrant is recursively subdivided.

In IP simulation we used a mix of artificially created terrain and real terrain taken from a Geographical Information System [6]. In both cases the input terrain forms a grid of points at regular 50 meter intervals in both x and y coordinates. The overall size of the terrain is 100 by 100 grid points.

3.3 Terrain Elevation Evaluation

After the quadtree is constructed, terrain elevations can be determined for any (x, y) coordinate point by searching the quadtree. Function *Get_Elevation* performs this lookup. It takes an (x, y) coordinate as input and returns the elevation (z value) at this location. This function must traverse the quadtree from the root to the appropriate leaf node which represents the area of terrain being examined. The x and y coordinates are substituted into the plane equation formed by the A, B, and C coefficients, stored in the leaf node, to determine the associated z value. For a square area of terrain consisting of n^2 points, function *Get_Elevation* has a worst case computation time of $O(log_4 n^2)$.

4. Terrain Avoidance

This section details the specific modifications to existing IP implementation required for IP to perform terrain avoidance. Specific references will be made to concepts, terminology, and algorithms defined in [1] and [9].

Consider a one-on-one air combat scenario with two players, *RED* and *BLUE*. Without loss of generality the *BLUE* player is IP. A lookahead computation for the *BLUE* player has two distinct phases. First trajectory trees are constructed for each of the two players. Secondly, the game tree used to select the next maneuver for the *BLUE* player is computed. Function *Lookahead* performs this computation.

```
Lookahead(BLUE,RED:state);

const T = 1.0;
      τ = 0.1;
      max_ply = 3;

var B_traj,R_traj:trajectory_tree;
      score:float;
      m:maneuver;

begin
  Compute_BLUE_Trajectories(BLUE,1.5,
                            B_traj);
  Compute_RED_Trajectories(RED,1.5,
                            R_traj);
  Select_Maneuver(B_traj, R_traj,
                  max_ply, 0,
                  score,m);
  return(m)
end;
```

During construction of the trajectory tree for each player, each new trajectory is compared with the underlying terrain to determine if the player has crashed. If a crash is detected then boolean variable *crash* is set to *true* in the corresponding trajectory tree node. This node of the trajectory tree becomes a leaf. In order keep the computation of trajectories isolated from the rest of the lookahead computation, the modifications to the code were isolated in the *Compute_Red_Trajectory_Tree* and *Compute_Trajectory* procedures. This choice allows the *Maneuver_Interpret* function to remain as an independent functional module from the rest of the IP implementation. The following modified *Compute_Trajectory* procedure illustrates the specific changes required to determine crashes. A similar modification was made to the *Compute_Red_Trajectory_Tree* procedure.

```
Compute_Trajectory(s:state; depth:integer;
                   var traj:trajectory_tree);
  var m:maneuver;
begin
  if depth = 0.5 then
    Maneuver_Interpret(s,m,τ,T/2,traj);
    traj.crash = Check_Crash(traj,T/2);
  else
    if depth ≠ 0 then
      Maneuver_Interpret(s,m,τ,T,traj);
      traj.crash = Check_Crash(traj,T);
      if not traj.crash then
        for m = SF to DL do
          Compute_Trajectory(traj[T/τ],
                depth − 1,
                Next_Trajectory(traj,m));
        end for;
      end if;
    end if;
  end if;
end;
```

Function *Check_Crash* takes a trajectory tree node as input along with the information as to which half of the trajectory is being evaluated. It returns a boolean which is *true* if the player has crashed into the terrain and *false* otherwise.

```
Check_Crash(var traj:trajectory_tree
                   T : float);
  var i:integer;
    elevation:float;
    crash:boolean;
begin
  crash = false;
  i = 1;
  while (not crash) and (i ≠ T) do
    elevation = Get_Elevation(traj[i].x,
                traj[i].y);
    if traj[i].z ≤ elevation then
      crash = true;
    end if;
    i = i + 1;
  end while;
  return(crash);
end;
```

The *Select_Maneuver* procedure drives the construction of the lookahead game tree. The selected maneuver is determined by a score that is propagated from the leaves to the root and takes the

probability of kill computed along the path from leaf to root during the lookahead computation into consideration. The scoring function is a reflection of the possible tactical situations. A score of -1 indicates that the RED player has the best possible tactical position. A score of $+1$ indicates that the $BLUE$ player is in the best possible tactical position. If a crash occurs then the associated game tree node becomes a leaf and the appropriate node score is assigned to it. Since a crash for the $BLUE$ player reflects the worst possible tactical situation, the lookahead computation is be forced (by the scoring function) to choose a maneuver that does not lead to a crash (if one exists) in the foreseeable future (the period of time that lookahead is performed).

```
Select_Maneuver(B_traj, R_traj:trajectory;
                max_ply, ply:integer;
                var score:float;
                var m:maneuver);

  var man,best_m:maneuver;
    child_score,SS,best_score:float;
    K_b,K_r,P_2,P_r,P_b:float;

begin
  if ply = 0 then
    P_2 = 1;
    P_b = 0;
    P_r = 0;
  else
    if odd(ply) then
      B_traj = Next_Trajectory(B_traj,m);
    else
      R_traj = Next_Trajectory(R_traj,m);
    end if;
    if not(B_traj.crash or R_traj.crash) then
      Probability_of_Kill(B_traj,R_traj,
                          ply, K_b, K_r);
      Compute_Game_States(K_b,K_r,P_2
                          P_r,P_b);
    end if;
  end if;
  if B_traj.crash and not R_traj.crash then
    score = -1
  else if R_traj.crash and not B_traj.crash then
    score = 1;
  else if B_traj.crash and R_traj.crash then
    score = -1;
  else if ply = max_ply then
    if odd(ply) then
      SS = Static_Score(B_traj[T/2τ],R_traj[T/τ]);
    else
      SS = Static_Score(B_traj[T/τ],R_traj[T/2τ]);
    end if;
    score = P_b - P_r + P_2 SS;
  else
    for man = SF to DL do
      best_score = -1;
      Select_Maneuver(B_traj, R_traj,
                      max_ply, ply + 1,
                      child_score,man);
      if child_score ≥ best_score then
        best_score = child_score;
        best_m = man;
      end if;
    end for;
    score = P_b - P_r + P_2 best_score;
    m = best_m;
  end if ;
end
```

The remaining module that requires modification to accommodate terrain is the $Check_Kill$ procedure. This procedure determines if either player is in the gun envelope of the other. The gun envelope defines a lethal area in which an opponent may be fired upon. However, if terrain blocks the line of sight between the two players then firing is disallowed. Thus, $Check_Kill$ must determine if the terrain blocks the line of sight between the players. The function $LOS_Blocked$ returns a boolean value of $true$ if the line of sight between the two players is blocked by terrain and $false$ otherwise.

```
Check_Kill(B_state,R_state:state;
                var B_kill,R_kill:boolean);
  var R_mgntd,B_mgntd,los_mgntd:float;
    LOS:vector;
    cos_α,cos_β:float;
    blocked:boolean;
begin
  B_kill = FALSE;
  R_kill = FALSE;
  LOS =line of site vector;
  LOS_mgntd =magnitude los vector;
  blocked = LOS_Blocked(B_state, R_state,
                LOS);
  if not blocked then
    R_mgntd =magnitude RED velocity vector;
    B_mgntd =magnitude BLUE velocity vector;
    cos_α = (R_vel · LOS)/R_mgntdLOS_mgntd;
    cos_β = (B_vel · LOS)/B_mgntdLOS_mgntd;
    if los_mgntd ≤ gun_range then
      if cos_α ≥ cos(gun_angle) then
        R_kill = TRUE
      end if;
      if cos_β ≥ cos(gun_angle) then
```

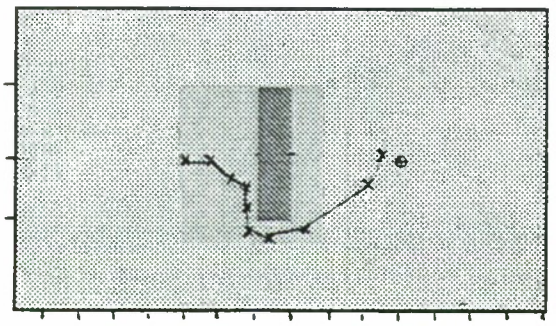


Figure 1: IP terrain avoidance example 1.

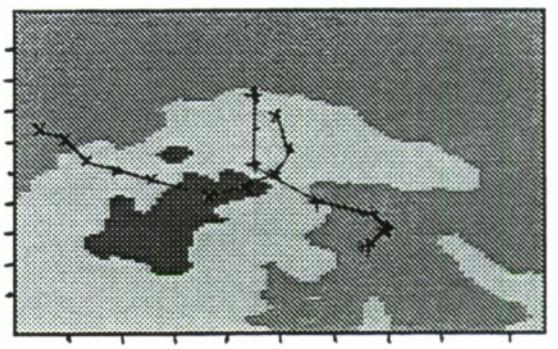


Figure 2: IP terrain avoidance example 2.

```
        B_kill = TRUE
      end if;
    end if;
  end if
end;
```

The *LOS_Blocked* function takes as input the states of both players and the line of sight vector. The function traces the line of sight vector through the quad tree to determine which quadrants the line of sight vector crosses. The function then determines if the line of sight vector intersects any of the involved quadrants by using their plane equations. At present this function is implemented using a rather brute force technique that is not particularly efficient and will not be further described.

5. Results

5.1 IP Behavior

Figures 1 and 2 show IP simulations involving terrain. Figure 1 illustrates an artificially constructed terrain region with a large obstruction. The *RED* player is placed in a stationary position to the east of the obstacle and facing east. The *BLUE* player is placed on the west side of the obstacle facing east. As the figure illustrates, the *BLUE* player is able to avoid the obstacle and assume a favorable tactical position on the *RED* player.

Figure 2 is a combat scenario that uses real terrain generated from the Geographical Information System. As the simulation verifies, both players are able to avoid the terrain while also exhibiting aggressive and tenacious behavior.

5.2 Execution Speed

Timing experiments on IP lookahead are designed to measure the elapsed execution time of IP for a specified number of plies. For each of the measurements reported lookahead was executed for the given conditions 100 times. The elapsed time of the total run was divided by 100 to get the average elapsed time for a single lookahead.

There are two cases to consider when measuring elapsed time of IP lookahead computation. If the players are in each others gun envelope, then the *Check_Kill* computation must be performed for each combination of players states. This adds considerable expense to the lookahead computation. Table 1 shows the elapsed execution times of IP lookahead when players are out of range of one another. Table 2 shows the elapsed execution times of lookahead when the players are in range of each other.

6. Conclusions

All simulation experiments of IP using terrain avoidance have demonstrated that IP is a tenacious and aggressive simulation entity. Execution

	Number of Maneuvers		
Plies	**5**	**7**	**11**
1	0.002	0.003	0.010
2	0.006	0.010	0.019
3	0.020	0.053	0.166
4	0.092	0.307	1.620
5	0.411	1.970	
6	1.800		

Table 1: IP execution time in seconds on IBM RS6000 when players are out of range.

	Number of Maneuvers		
Plies	**5**	**7**	**11**
1	0.003	0.006	0.010
2	0.008	0.022	0.030
3	0.026	0.183	0.310
4	0.133	1.620	3.137
5	0.456		
6	1.950		

Table 2: IP execution time in seconds on IBM RS6000 when the players are in range.

timing results demonstrate that IP is capable of 3-ply lookahead in under one second on IBM RS6000 workstation. Thus, IP is still capable of planning even when terrain avoidance is performed.

It is important to note that these execution speeds for IP should be considered worst case since quadtrees do not provide a particularly efficient lookup function. We expect that a 486 class PC or IBM RS6000 class workstation with sufficient memory to support the use of polygon terrain representation or digital terrain models would significantly improve execution speeds of lookahead computation.

A more efficient computation for the *LOS_Blocked* function should also improve execution speeds. Future efforts should investigate improving this computation by using a method akin to Bresenham's line drawing algorithm [10] to trace through the quadtree along the line of sight vector. This should significantly reduce the total number of quadrants which must be searched during this computation.

7. References

[1] Amnon Katz, "Intelligent Player - First Principle Foundations", *Proceedings of Third Computer Generated Forces Conference*, 1993.

[2] Amnon Katz and A. Ross, "One on One Helicopter Combat Simulated by Chess Type Lookahead", *Journal of Aircraft*, 28, no. 2, 1991.

[3] Amnon Katz and Bret Butler, "A Flight Model for Unmanned Simulated Helicopters", *Journal of Aircraft*, 29, No. 4, 1992.

[4] Amnon Katz, Bret Butler, and D. Allan, "A Computer Generated Helicopter for Air to Air Combat", *AIAA Simulation Technologies Conference*, New Orleans, 1991.

[5] Ellis Horowitz and Sartaj Sahni, "Fundamentals of Computer Algorithms", *Computer Science Press*, 1978.

[6] David J Maguire, Michael Goodchild, and David Rhines, "Geographical Information Systems Principles and Applications", Longman Scientific and Technical, 1991.

[7] F. Austin, G. Carbone, M. Falco, and H. Hinz, "Automated Maneuvering Decisions for Air to Air Combat", Grumman Report RE- 742, November 1987.

[8] Michael D. Petty, "Terrain Reasoning for Reconnaissance Planning in Polygon Terrain with Cultural Features", Institute for Simulation and Training, IST-TR-93-03, Jan 1993.

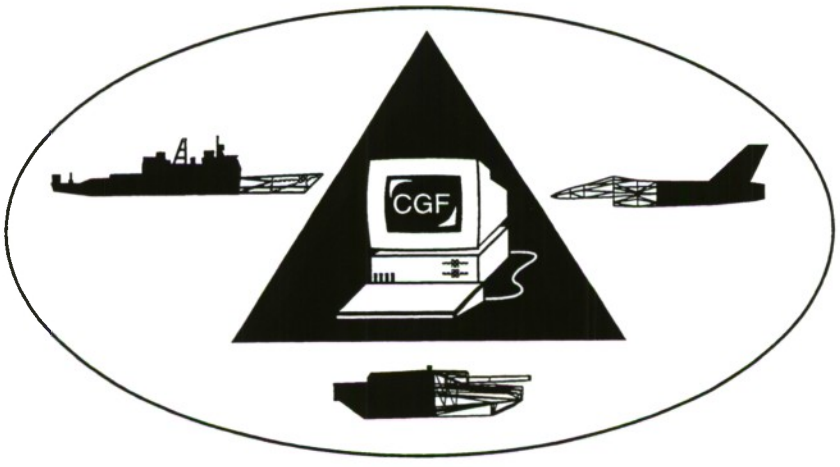
[9] Gregory A. Schaper, Shridhar Pandari, and Mandeep Singh, "Lookahead Limits of Intelligent Player", Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, Institute for Simulation and Training, Orlando, FL., 1994.

[10] J.D. Foley and A. Van Dam, "Fundamentals of Interactive Computer Graphics", Addison Wesley, 1982.

8. Author's Biographies

Gregory A. Schaper. Currently an Assistant Professor in the Department of Computer and Information Sciences at East Tennessee State University. Earned PH.D. from University of Central Florida in Computer Science in 1989. Received Bachelors degree in Computer Science from Arkansas State University in 1984. Has worked as a Research Associate at the Institute for Simulation and Training, Orlando, FL (1987-1988) and McDonnell Douglas Helicopter - Simulation Division, Phoenix, AZ (1990). Areas of interest include Computer Architecture, Parallel Processing, Graph Theory, Battlefield Simulation, and Neural Networks.

Ashok Pandari. Currently a graduate student in the Department of Computer and Information Sciences at East Tennessee State University. Earned B.S in Computer Science and Engineering from Jawaharlal Nehru Technological University, India, in 1993. Is a Graduate Assistant in Department of Housing, East Tennessee State University. Areas of interest include Database design, Graphical User Interfaces and Object Oriented Programming.


CGF

Recent Developments in ModSAF Terrain Representation

Joshua E. Smith
Loral Advanced Distributed Simulation
80 Pleasant St., Barre, MA 01005
jesmith@camb-lads.loral.com

1. Abstract

After five years of stability in representation, the growing demands of terrain complexity and terrain reasoning within the DIS environment have forced a re-engineering of terrain representation within ModSAF. The representations inherited from the SIMNET and Odin legacy have been changed significantly to support current and future terrain requirements. This paper will discuss the most significant changes, and will detail the implications of these developments for the rest of the CGF community.

2. Terrain Representation Legacy

The terrain representation currently used in ModSAF is derived from a legacy of prior representations, as shown in Figure 1.

2.1 LibTDB

The terrain representation used in SIMNET SAF was a format called `libTDB`. This was a completely polygonal format. For example, the only representation of roads in this format was as a sequence of triangles and rectangles which composed the surface of the road. This representation was completely interoperable with SIMNET simulators,

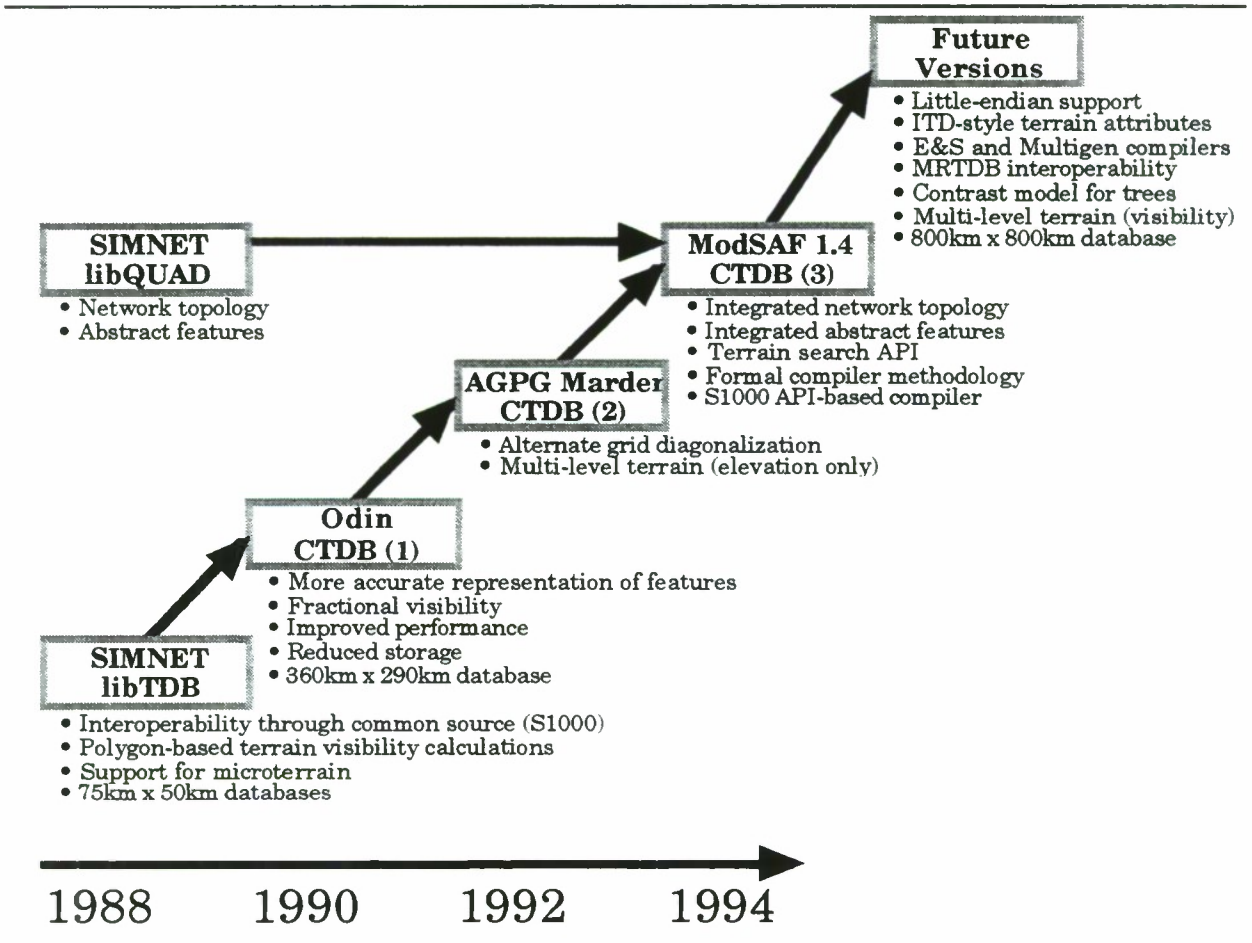


Figure 1: ModSAF Terrain Legacy

because it was based on the same source data.

2.2 LibQuad

In addition to this format, another terrain representation called `libQuad` was also used in SIMNET. This representation was designed to support map display and planning. It included a network topology which could be used to plan road routes, breach river obstacles, etc. This format also included "abstract" terrain features, such as lake and forest footprints.

2.3 CTDB Format 1

The `libTDB` format worked well with SIMNET-size databases which were typically less than 100 km on a side, but proved problematic for very large databases, such as the 290 km by 360 km "SAKI" (Saudi Arabia, Kuwait, and Iraq) database. To support this database, and to improve performance on smaller databases, a new format called CTDB, for **C**ompact **T**errain **D**ata**B**ase, was developed.

The CTDB format took advantage of the gridded nature of SIMNET terrain databases to significantly reduce storage requirements. It also used various real time compression mechanisms (such as fixed point numeric representations) to reduce storage demands. Reducing size improves performance in two ways. First, more of the terrain area can be brought into active memory, reducing the need for disk access. Second, by squeezing more geography into fewer bytes, data cache coherency is improved, which can lead to performance gains on RISC processors.

In addition to these improvements, new algorithms were developed for performing visibility calculations, determining ground elevations, placing entities on the terrain, and other operations. These algorithms both improved performance, and in many cases increased fidelity. For example, the time to complete a visibility calculations was reduced by 75%, while the results were changed from a discrete (visible, partially visible, invisible) to an analog result (visible area).

It was around the same time that ModSAF development was started in support of the DARPA WISSARD program, under the direction of CDR Dennis McBride. The ModSAF system used the CTDB and `libQuad` terrain representations without modification.

2.4 CTDB Format 2

In 1992, the ModSAF program was adopted for use as the Computer Generated Force simulation in the German AGPG program (a platoon level troop training system for the Marder Infantry Fighting Vehicle). The most significant additions made under this program were the introduction of an alternate diagonalization of the terrain grid, and "multi-level" terrain.

2.4.1 Alternate Diagonalization

The SIMNET databases all used a northwest to southeast diagonalization for the regular grid of elevations. For interoperability with its image generator, the AGPG program required that the diagonalization instead be northeast to southwest. The CTDB database format was modified to support either diagonalization, and the various terrain analysis algorithms were modified to support both diagonalization options. The specification of diagonal direction was made on a per-database basis.

2.4.2 "Multi-Level" Terrain

One of the ways the performance of terrain analysis algorithms can be improved is by introducing "implicit" information in the representation. For example, if one can assume that it is impossible to see *under* a terrain polygon, then a visibility calculation can be performed which only checks polygon edges – if line of sight passes under a polygon edge, then visibility is necessarily blocked.

However, if one introduces multiple levels of terrain (tunnels, bridges, etc.), then this implicit information is not correct. Thus, the algorithms for visibility calculation need to be generalized to remove this assumption. Of course, this would have a detrimental impact on performance. The compromise struck in the CTDB representation is to divide terrain into *classes*. In addition to the regular grid, which provides general coverage of the terrain area, portions of the terrain surface can be represented by:

- **Base Terrain**: A Triangular Irregular Network (TIN) which abides by the implicit assumption that visibility under the triangles is impossible.
- **Multi-Level Terrain**: A TIN which does not abide by this assumption.
- **Default Terrain**: A TIN which should be used only if no other terrain triangles are found at a location.

The last type was introduced to allow compact storage of areas where the regular grid was replaced only *partially* by a TINned region (such as a river bed).

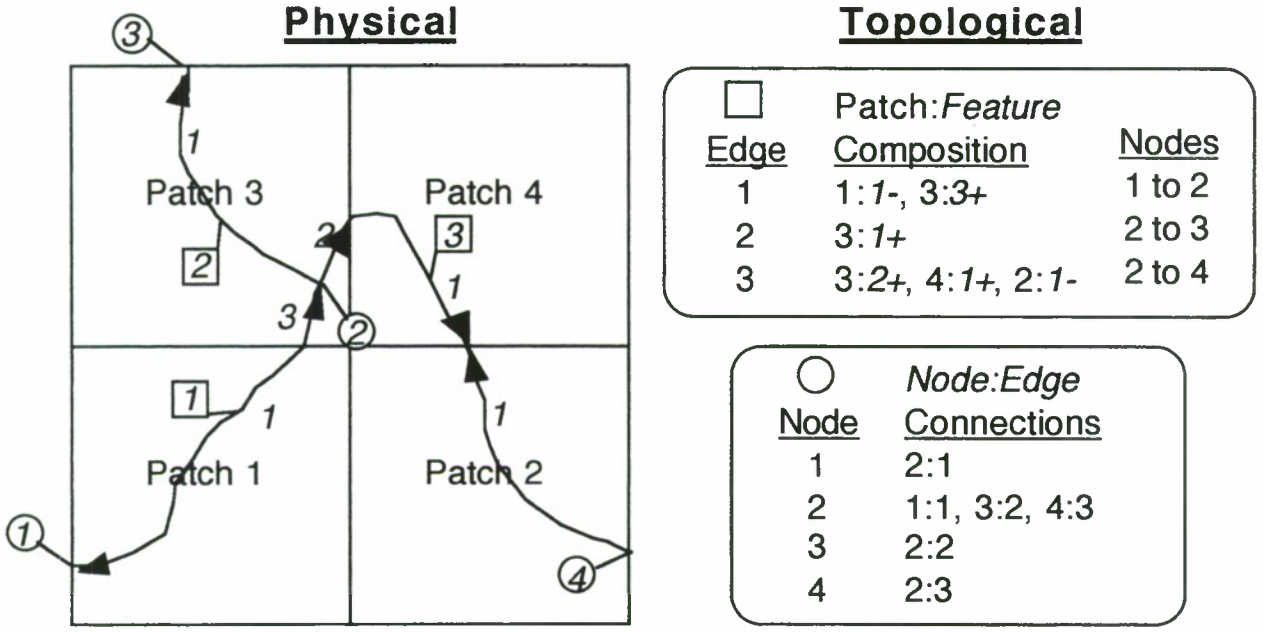


Figure 2: Integrated Network Topology

The implementation of multi-level terrain for AGPG was only preliminary. A mechanism was added to the file format to represent this information, and the terrain elevation lookup algorithms were modified to support fetching elevations using a *reference elevation* which is used to determine what level of terrain should be used. For visibility calculations, the multi-level terrain polygons are simply ignored (as if all bridges are made of glass).

3. CTDB Format 3

Although the `libQuad` database had served the needs of ModSAF well through version 1.3, it had some drawbacks which needed to be remedied. The primary problem was the terrain compiler, which was rather fragile and only worked on an outdated computer platform. In addition, the format and software were developed before the ModSAF software quality and documentation standards had been adopted. As such, it was quite difficult to maintain.

Another nagging problem with the `libQuad` format was its memory consumption. The entire database had to be read into memory (no caching was supported), and much of the information was redundant with data stored in the CTDB format. Finally, the `libQuad` software did not provide much of an Application Programming Interface. To traverse features in the `libQuad` database, an application would have to traverse the actual data structures used internally by `libQuad`. (Note that CTDB also suffered from the same problem.)

To alleviate these problems, LEADS[1] funded the development of CTDB format 3

3.1 Integrated Network Topology

CTDB now supports a complete integrated network topology, as shown in Figure 2.

The physical representation of each linear feature (road or river) is stored with other physical information in the CTDB "Patch" data structure. The topology makes reference to this information, without duplication. For example, in the figure, edge 1 consists of two linear features: linear feature 1 of patch 1 traversed last-to-first (1:*1-*); and linear feature 3 of patch 3 traversed first-to-last (3:*3+*).

[1] Software development was funded by LEADS (Loral Experimental and Developmental Simulation System), a Loral corporate chartered project to promote the effective use of DIS throughout Loral. However, this software is being provided to the government with exactly the same rights as other ModSAF software so that it can be freely used throughout the DIS community.

The physical representations of the linear features in turn reference the topologic edge. For example, feature 1 of patch 1 and feature 3 of patch 3 both reference edge 1. This is important, because the physical representation is stored using *spatial indexing*, which means that given a location, a feature can be found quickly. Then, given the physical feature at a point, a planning algorithm can go directly to the topology of the area without search.

3.2 Integrated "Abstract" Features

As explained earlier, the `libQuad` database included "abstract" features. With the elimination of `libQuad`, CTDB is required to pick up these features. The spatial indexing used for physical features (a grid of small patches) is not particularly good for representing these terrain abstractions because forested areas and lakes tend to cover large areas. The natural data structure to use for these is a dynamic quad tree, as shown in Figure 3.

The file format allows nodes of the quad tree to be expanded using any criteria. In the example above, the distribution of features warrants smaller quad nodes in some places, and larger nodes in others. Features are stored in both interior and leaf nodes of the quad tree, to achieve fairly even distribution.

These features abstractions are not actively used by the terrain analysis algorithms within the CTDB software (visibility calculations, elevation lookup, etc.). CTDB acts only as a repository for this information. This means that the format can be quite flexible about what sorts of abstractions are stored. Currently, CTDB has enumerations and instance data defined for many abstract features, as shown in Figure 4. Of course, each feature also has a series of locations which define its position, extents, or content. This list can be easily extended to support additional abstract feature classes.

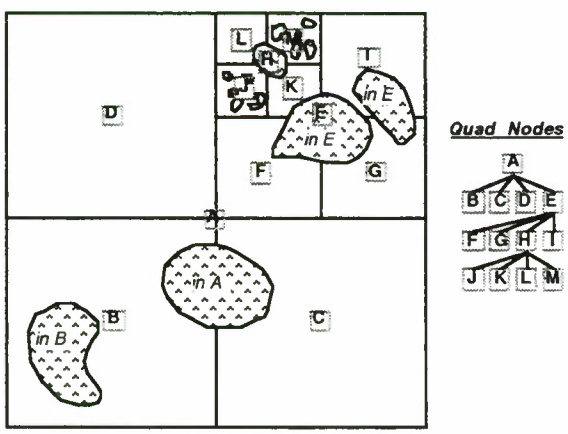


Figure 3: Quad tree of features

Feature	*Description*	*Instance Data*
Quad Node	Indicates that this is an interior quad node, and that the children of this node are represented elsewhere in the list of abstract features.	*Child node storage locations*
Canopy	Tree canopy footprint.	*Penetrability*
Soil Defrag-mentation	Bounding footprint of an area with a uniform soil type, such as a lake or a marsh.	*Soil type Layer number*
Steep Slope	Bounding footprint of an area with steep slope.	*Slope*
Railroad	Railroad lines.	
Pipeline	Pipeline lines.	
Political Boundary	Political boundaries, such as country borders, or city limits.	
Label	Map labels, such as town names.	*Text*
Tactical Sign	Areas identified as tactically dangerous.	
Off-road Segment	Precomputed traffic networks indicating desired routes which do not follow physical linear features.	*Topological edge*

Figure 4: Abstract Feature Enumerations

3.3 Terrain Search API

With the centralization of all terrain information within a single software module, it became clear that a uniform application programming interface (API) was needed to support access to this data. The interface paradigm chosen is one quite common in this domain: iterative fetching. Pseudocode demonstrating this paradigm is shown in Figure 5.

```
Global Search
Create a Search Space (Geographic Extents, Iteration Options)
Repeat for Each Type of Feature:
    Repeat until No Features of a Type Remain in the Space:
        Get the Next Feature of a Type (Search Space)
        Process the Feature
Destroy the Search Space (Search Space)

Local Search
Create a Search Space (Geographic Extents, Iteration Options)
Repeat until No "Patches" Remain in the Space:
    Get the Next Patch (Search Space)
    Repeat for Each Type of Feature
        Repeat until No Features of a Type Remain in the Patch:
            Get the Next Feature of a Type (Patch)
            Process the Feature
Destroy the Search Space (Search Space)
```

Figure 5: Iterative Fetching Paradigm

To access terrain features, an application creates a search space which is defined to contain all the features in a geographic area. The application can then retrieve these one at a time from the space. As the figure demonstrates, two search methods are supported: *global* search and *local* search. Global search is simpler to use, but fails to exploit the spatial indexing of the system. In a local search, the application is given all the features from the search area, but they are returned in an order which is more natural for the internal representation of the data. Thus, an application which uses the local search approach will typically execute faster than one which uses the global approach.

4. Terrain Compilers

With ModSAF version 1.4 the mechanisms used to "compile" CTDB databases from source formats have been solidified and documented. The ModSAF distribution now includes the source code to build two terrain compilers: an S1000 to CTDB compiler, and a CTDB to CTDB recompiler (for converting between CTDB formats, and correcting errors in existing CTDB databases). Also included in the distribution are software and documentation which can be used to construct new terrain compilers for other source formats.

The compiler is divided into two components: the *back end* which builds the CTDB data structures and the *front end* which assembles the required data.

4.1 Compiler Back End

The compiler back end provides functions to assist in the encoding of terrain data, so that it can be stored in a CTDB format terrain database file. The compiler back end does not change the content of the data given to it – only the format. The back end does perform the following data scrubbing, which does not change the nature of the physical representation, but will reduce storage space:

- Clipping of buildings, tree lines, and linear features at patch boundaries.
- Elimination of vertical terrain or tree canopy triangles.
- Elimination of intermediate 3D collinear vertices in tree lines, and 2D collinear vertices in linear features.
- Elimination of repeated vertices in features.
- "Defragmentation" of connected linear features (identifying that two adjacent linear features have identical attributes, and thus can be represented as a single linear feature).
- "Fragmentation" of intersecting linear features (except when the intersection occurs in the bounds of a multi-level terrain polygon, where linear features may actually cross without intersecting).
- Elimination of duplicate linear segments.

The back end of the compiler also has functions which can generate the network topology from the physical and abstract linear features. This is required because few source formats have a complete integrated topology already computed. Note that this is an error-prone process, however, and should be considered a stop-gap measure as we wait for source formats to provide the complete picture of the terrain.

4.2 Compiler Front Ends

The compiler front end is responsible for retrieving terrain information from a source format, and providing it to the back end for encoding. The front end is responsible for execution flow (it is the "`main`" program), and file I/O operations. The front end performs the steps shown in Figure 6.

1. Open a file for output.
2. Fill in the file header.
3. Unpack the header into a CTDB structure (the CTDB software archive provides a utility function to do this).
4. Generate a list of physical features in memory. This is done by making repeated calls to a feature encoding function. Put the feature count into the header.
5. Ask the compiler back end to derive a caching scheme so that terrain can be efficiently accessed at run time. Put the result into the header.
6. Generate the quad tree of abstract features. Again, this is done by making repeated calls to a set of feature encoding functions. Put the count into the header.
7. Generate the lists of nodes and edges (the network topology) in memory. This can be done by calling a function in the back end which regenerates topology. Put the sizes and counts into the header.
8. Write the header to the output file.
9. Write the physical features to the output file.
10. Write the grid of elevations and feature-presence flags to the output file.
11. Write the nodes and edges to the output file.
12. Write the abstract feature quad tree to the output file.
13. Close the output file.

Figure 6: Steps Performed by the Front End

To simplify the task of writing front ends, the ModSAF 1.4 software distribution includes a CTDB to CTDB "recompiler". This program uses CTDB as

its input format, and follows all the steps of a standard terrain compiler. Since all the data is already formatted exactly as needed for encoding, the compiler is very simple. It can be used as a template for developing new compilers.

5. Future Directions

CTDB is not a "dead" format. It continues to be adapted as the complexity, size, and nature of the environmental representation within DIS evolves. The following sections describe some of the changes which will likely be added to CTDB in the very near future.

5.1 MRTDB Interoperability

The CGF component of the CCTT program will be using a new terrain format called MRTDB. This format is much like CTDB in functionality, but is using new approaches in spatial indexing, model storage, and surface representation. In order to interoperate with CCTT CGFs and simulators, the CTDB format will need to be upgraded to include some of the more significant features of MRTDB.

5.1.1 Variable Diagonalization

The MRTDB format will support varying grid diagonalization on a per-grid-cell basis, instead of just per-database. This allows the terrain representation to more closely match reality, with significantly less overhead than using arbitrary triangles (often referred to as "microterrain"). While CTDB *could* interoperate with this terrain by using a TIN, performance will be enhanced if CTDB is instead modified to also support varying diagonalizations.

This will most likely be implemented by taking one bit of the elevation data currently stored for each grid cell, and instead using that bit to indicate which diagonalization is being used (⧅ or ⧄). This will result the loss of either a bit of resolution or a bit of representable range in elevations.

5.1.2 Tree Model References

Another innovation in the MRTDB format is the use of model references. The idea is that rather than storing all the information about an object at a location, a *reference* to an object is stored. Each unique object is placed in a library, and substitutions of objects for references are made on the fly. While this is not helpful for buildings in CTDB (which are each uniquely placed on the terrain), the idea is quite handy for trees, which have the potential for a lot of commonality between instances.

The CTDB format will likely be changed to use the model reference approach for trees. In place of the radius of each tree (which is currently stored as a 16 bit fixed point number), an index into a *tree table* will be stored. Each entry in the tree table will have:

- Foliage radius
- Trunk radius
- Foliage opacity for visual sensors, thermal sensors, etc.

5.2 Tree Contrast Model

When CTDB was originally defined, the "right" way to use trees in visibility calculations was unknown. A guess was made that it was reasonable to use tree transmittance to effectively *decrease the size* of an object being viewed. It turns out that this is not the best way to model loss of visibility due to trees. Instead, the cumulative opacity of all trees intersected should be returned separately from the apparent size of the object, so that they may be used to determine *loss of visual contrast*, a parameter of the Night Vision Laboratory target detection model used in ModSAF (Courtemanche et. al. 1994).

Also, the responsibility for knowing what tree opacity values are will be transitioned from the application to the terrain representation. The application will merely specify a sensor type, and this will be used to select a tree opacity for each tree encountered. This will allow more variation in the types of trees represented within CTDB.

5.3 Mobility Information

The representation used for mobility in CTDB is a holdover from the SIMNET databases. Each polygon is assigned a numeric "soil type" value which can be used by an application to control platform mobility. This approach is insufficient to represent the wide variety of mobility types desired on modern databases. Furthermore, there are no validated kinematics or dynamics models available which can use this generalized information.

Thus, CTDB is likely to be changed to support tables of mobility characteristics similar to the information available in DMA ITD data sets. The individual polygons of the database will hold indices into these tables. For backward compatibility to existing simulations which use CTDB, one of the entries in the mobility table should be a mapping to the SIMNET constant most appropriate for the area.

5.4 Visibility on Multi-Level Terrain

As mentioned earlier, multi-level terrain can be represented in CTDB, but not all the algorithms have been updated to use this terrain correctly. Specifically, the visibility calculations do not account for blockage due to multi-level terrain (a platform cannot hide under a bridge, for example). These algorithms will be updated to support multi-level terrain fully.

5.5 Larger Databases

The largest CTDB database compiled to date is 416 km by 416 km. Without modification, CTDB can conceivably handle databases as large as 800 km on a side. Beyond that size, local coordinates are insufficient (UTM projection becomes invalid over larger ranges, and the loss of accuracy in local tangent coordinates makes them insufficient as well). Thus, for larger areas, a *tiling* approach will be needed. The terrain surface will be represented by tiles which each have their own local coordinate system, and a translation mechanism will convert between that and a system of global coordinates.

5.6 Little-Endian Support

The current CTDB format and software assume execution on a "big-endian" byte order platform (SGI, Sun, Motorola, etc.). Some modification to the software will be necessary to support "little-endian" CPUs (DEC Alpha, Intel), but we anticipate making these extensions in the near future. In preparation for this change, the naming convention used for CTDB databases has been changed, as shown in Figure 7.

Extension	*Byte Order*
`.ctb`	Compact Terrain Big-Endian
`.ctl`	Compact Terrain Little-Endian

Figure 7: File Naming Convention

5.7 New Source Formats

Currently, CTDB compilers are available to convert from S1000 (the format used for the large SIMNET repository of databases), and from CTDB. A compiler was also written to convert from the interchange format used on the AGPG training system mentioned earlier.

Efforts are currently underway to create compilers which convert Evans & Sutherland and Multigen formats to CTDB. Also, for applications where interoperability with manned simulators is not required, it may be possible to convert directly from DMA products such as DTED and ITD. We hope and expect that ModSAF users will undertake conversion efforts using the compilation tools provided with the ModSAF software tools, and will share these with the community at large.

6. Acknowledgments

This work is being supported in part by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058, and in part by LEADS, a Loral corporate development program.

7. References

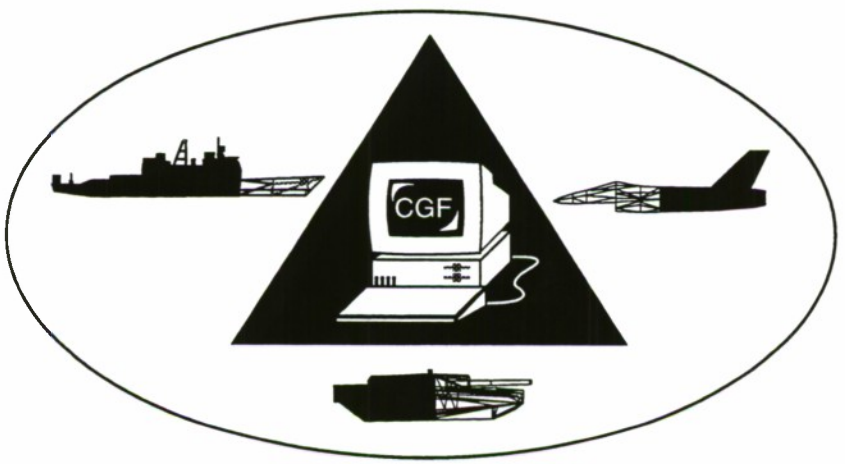
Courtemanche, A. J., Monday, P., (1994). "The Incorporation of Validated Combat Models into ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, University of Central Florida.

Smith, J. E. (1994). *LibCTDB: Compact Terrain DataBase Library User Manual and Report*, Loral Advanced Distributed Simulation, Cambridge, Massachusetts.

Stanzione, T., Smith, J. E., Brock, D. L., Mar, J. M. F., Calder, R. B. (1993). "Terrain Reasoning in the ODIN Semi-Automated Forces System", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, University of Central Florida.

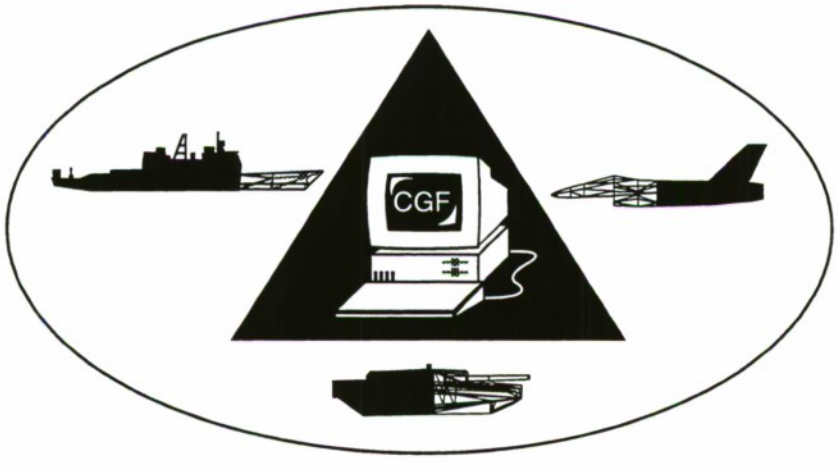
8. Author's Biography

Since graduating from Worcester Polytechnic Institute with a Bachelor of Science in Computer Science in 1988, Mr. Smith has been working in the Semi-Automated Forces group of Loral Advanced Distributed Simulation (formerly BBN Advanced Simulation). He is a Software Engineering Specialist with Loral, responsible for the software architecture of ModSAF. Mr. Smith is the chair of the Computer Generated Forces working group of the DIS standards workshop.


CGF

Session 8a: Applications of CGF

Metzler, LB&M Associates Inc.
O'Keefe, U.S. Army, Natick RD&E Center


CGF

A Method to Quantify the Application Value of Intelligent Decision Support Systems

Theodore Metzler and Joseph Kelly
LB&M Associates, Inc.
211 SW A Ave.
Lawton, OK 73501-4051
metzlert@lbm.com kellyj@lbm.com

1. Abstract

Published methodologies for evaluating Intelligent Decision Support Systems (IDSSs) have tended to neglect the important need to quantify their organizational impacts--i.e., their *applied* value. We present the outline of an approach that directly addresses this need, recommending use of the Distributed Interactive Simulation (DIS) infrastructure, exploitation of existing software resources that support "intelligent objects" and application of a simple Value Calculus for data analysis.

2. Introduction

Intelligent Decision Support Systems (IDSSs) offer assistance to decision makers on a range of application areas that includes business management, medicine, and military operations. The advanced technologies typically incorporated in these systems (such as neural networks, expert systems, Dempster-Shafer evidential reasoning, fuzzy logic and the like) tend to introduce substantial development costs. Moreover, performance of the systems can have important practical consequences. With business applications, for example, significant sums of money may be at stake. Medical applications, as Wyatt and Spiegelhalter have observed (Wyatt 1990), present the IDSS also with ethical responsibilities--a condition obviously shared by military applications, in which the lives of our soldiers may be the cost of deficient performance. Hence, it is reasonable to take seriously--even during early stages of research and development--the task of quantifying the value of introducing IDSSs to given applications.

Unfortunately, progress with this task in the Artificial Intelligence (AI) community has generally left much to be desired. As Cohen and Howe candidly acknowledge in AI Magazine, "... we rarely publish performance evaluations and, still less, evaluations of other research stages" (Cohen 1988). Again, Wyatt and Spiegelhalter point out that although "many believe that medical expert systems have great potential to improve health care, ... few of these systems have been rigorously evaluated ..."(Wyatt 1990). Similarly, the authors of the present paper recognize military leaders are still waiting to see the presumed value of IDSSs clearly quantified for battlefield applications.

This need for methodological advancement of evaluation appears also to be more urgent in some stages of the IDSS lifecycle than in others. Laboratory testing of expert systems, for example, has received relatively more attention, producing a number of useful techniques such as black-box / white-box testing; checks for the consistency, completeness and redundancy of rules; and measures for the accuracy and robustness of their conclusions (Wyatt 1990, Kirani 1992). Undoubtedly, these techniques contribute to determining the overall worth of an IDSS, for it certainly *matters* whether advice produced by the system is timely, accurate, reliable and so forth.

On the other hand, a conspicuously *less* advanced stage of evaluation methodology is noted in the following comment by Heathfield and Wyatt (Heathfield 1993) regarding medical IDSSs: "Many evaluation methodologies have addressed specific aspects of system structure and function, but have not addressed evaluation of the impact of the system on users or patients." Indeed, one may properly ask, "*Given* that this system produces medical advice that is timely, accurate and reliable, *how much* does its use in a *hospital setting* actually improve *health services*?" Again, the business executive may wonder, "*Given* that this IDSS is technically

sound, *how much* will my corporation benefit from its everyday *employment* in my *operations*?" In a similar fashion, it is reasonable for the military commander to enquire, "*Given* that this IDSS on my howitzers can produce advice that is timely, accurate, reliable, etc., *how much* will its introduction to my artillery *battalion* actually improve our performance of particular *missions*?" In each case, the most fully-developed evaluation methods now available tend to fall silent when this *type* of question is asked. If a name is required for the missing piece of methodology, perhaps we should say a general method is needed to quantify the *application* value of IDSSs.

A method to meet this need is described in this paper. Although our proposed solution is sufficiently generic to serve any of the kinds of application areas previously mentioned, we shall use the context of a specific U.S. Army Field Artillery program to illustrate its details.

3. Problem Statement

The U.S. Army requires a new generation of artillery vehicles, comprised of an improved howitzer and its companion resupply vehicle (collectively designated the Crusader system). The initial operational concept (Preliminary 1994) specifies this system must "enhance the capability of the Field Artillery to provide supporting fires" for maneuver forces equipped with modern armored vehicles. Provision of this enhanced capability will require a Crusader battalion to demonstrate a "significant increase" in the ability to accomplish assigned missions (such as deliver fires, communicate, move, survive, resupply and maintain). One of the difficult problems presented by this requirement is that of determining whether--and, quantitatively, to what degree--certain IDSS capabilities on Crusader vehicles will improve system performance. For example, Army planners need to know--at early stages of research and development--just *how much* better the Crusader howitzer and the howitzer platoons may be expected to accomplish "move" missions if they are furnished with an automated route-planning IDSS component. Questions of this kind are particularly challenging at the present time, since neither vehicle is available yet for testing (a type of problem not uncommon in other application areas as well). Accordingly, our solutions (at least, in the near-term) must often involve the use of *simulation*--and, in the given case, appropriate simulation should represent complete Crusader units in realistic operational scenarios.

4. Solution - Part 1: Simulation Environment

Fortunately, many of the resources for solving the problem we have described can be found in the infrastructure of Distributed Interactive Simulation (DIS). Evolving DIS architecture and protocol standards are envisioned as ultimately supporting a "wide spectrum of applications" (The DIS Vision 1994), recommending them as a potentially generic environment to serve the needs of such areas as business and medicine. Currently, they already furnish a framework for exercises representing the military operations of existing armor and artillery units. With the provision of additional crewstation simulators for Crusader vehicles--and the provision of selected prototype IDSSs--so-called "virtual" DIS exercises should be possible in the future to support evaluation at the level of individual soldier or section performance. In the near-term, however, other capabilities of the DIS infrastructure are likely to be more cost-effective. A particularly attractive option would be to exploit the existing technology of Computer Generated Forces (CGF) by adapting it to serve the problem we are addressing.

5. Solution - Part 2: Intelligent Objects

Generically, the CGF entities generated for a military DIS exercise (e.g., tanks, helicopters, etc.) may be characterized as (artificially) "intelligent objects." For example, a simulated tank created with ModSAF, the modular CGF software system developed by Loral (Ceranowicz 1993), can automatically perform a number of tasks (such as route planning or road following) that would normally be performed by a human operator. Moreover, the intelligent behavior of this object can be changed to simulate the effect of an IDSS upon task execution. One of the methods for implementing such changes in ModSAF is to alter the system parameters in appropriate task models (Mohn 1994). The ModSAF software package also permits human operators to direct the behavior

of CGF entities, selectively, offering an alternative mechanism for simulating IDSS effects upon vehicle behavior. That is to say, it is technically feasible to modify and extend existing ModSAF entities in ways that allow them to simulate the battlefield behavior of Crusader vehicles--with or without particular types of IDSS equipment. Moreover, the ModSAF environment includes extensive automated logging of data concerning behavior of CGF entities in a simulation exercise.

All of the basic resources, then, are available in DIS and environments such as ModSAF to support data collection for solution of the given problem. A representative experiment, using this approach, would involve comparative analysis of data collected from two simulation exercises (in the DIS environment and with Crusader CGF units generated by suitably adapted ModSAF components). Both exercises would employ a common battlefield scenario and Crusader units, but only one of them would simulate performance of the units equipped with a chosen IDSS component or system. The other exercise would yield "baseline" data for comparisons. A similar experimental procedure (albeit, without use of CGF entities) has been employed successfully in a simulation testbed to validate knowledge-based sensor control (Harrison 1994). Both DIS and the ModSAF technology happen currently to be focused upon military applications; however, the DIS protocol standards, as well as the modular "intelligent objects" of ModSAF, are clearly suitable resources for extension to other domains, such as business and medicine.

It may be objected that the proposed procedure is "circular"; i.e., provision in the CGF entities of simulated effects of IDSS equipment may appear to "beg the question." The objection, however, overlooks two important points. First, the proposed procedure must be understood as part of a larger, comprehensive evaluation process. Initial estimates of IDSS effects upon individual task performance for individual vehicles can later be refined by more complete (and more expensive) man-in-the-loop crewstation simulations. Second, even when such refinements become available, the proposed experiments will still be needed to determine the organizational impact of the (now more precise) quantifications of task or vehicle-level effects.

As previously noted, the form of the question we are addressing concerns the *application* value of an IDSS. *Given* certain assumptions about the accuracy, speed, etc., with which a particular IDSS component on individual vehicles allows specific tasks to be performed, we wish to explore systematically the consequences of these assumptions for *units* of such vehicles (e.g., Crusader platoons) executing higher-level *operations* (e.g., "move" missions) under the conditions of realistic *battlefield* scenarios. Knowing the rule base in a given IDSS is consistent may, indeed, be an important piece of evaluative information, but it does *not, ipso facto*, tell a Brigade or Division commander what the *applied* value of the IDSS will be for his Field Artillery battalion on the battlefield. To determine the value quantitatively, data must be collected in a realistic simulation of battlefield operations, and then subjected to analysis--a solution step we shall now address. The "Value Calculus" we propose for this step is based upon a simple utility model that has been applied successfully to prior evaluation tasks by one of the authors of this paper (McGee 1991).

6. Solution - Part 3: Value Calculus

Techniques for data analysis can be made arbitrarily complex, but no amount of mathematical labor can transform bad data into useful information. Accordingly, we acknowledge at once the importance of some preparatory work that is assumed in all applications of our Value Calculus.

First, it is necessary to identify primitive units and operations for which appropriate measures of performance can be defined. In the military problem we have selected, individual *vehicles* (e.g., Crusader howitzers or resupply vehicles) would illustrate elementary *units*, and their assigned *tasks* (e.g., plan route, follow route, etc.) would illustrate elementary *operations*. An example of appropriate measures of performance, in this context, might concern the merits of a route-planning task execution on an individual Crusader vehicle, and be comprised of specific measures such as the following:

⇒ speed with which the route is planned

⇒ degree to which the planned route avoids exposure to enemy detection

⇒ overall length (i.e., directness) of the planned route

We shall designate such measures *"merit factors,"* and note that their specification serves to define certain *data elements* that must be collected in each simulation exercise. Knowledge of these required data elements, will, in turn, play a role in construction of the battlefield *scenario* to be used in the simulation exercises; minimally, for example, the scenario for the case we are considering will need to task at least one Crusader vehicle with route planning. Only when preparatory work of this kind has been well performed can we expect the Value Calculus--or any other analytical technique, for that matter--to produce sound quantification of IDSS application value.

For each merit factor associated with a particular task and vehicle type, the Value Calculus maps raw data (often, averages of collected data elements) onto the real-valued interval [0,1] by means of a specific *satisfaction function*, producing a *merit value* for the given merit factor. For example, the average time in which an individual Crusader vehicle completes the route planning task in the course of an exercise may happen to be 3 minutes; in this case, an appropriate satisfaction function takes 3 as its argument and returns some value such as 0.75. In the next step, the Value Calculus determines a primitive *Figure of Merit* (FOM) for the given vehicle and task by computing a weighted sum, as shown in the following equation:

(1) $\text{FOM}(V(v_i), T(t_j)) = \sum_n w_n f_n(x)$

where

$V(v_i)$ = vehicle v_i (in this case, a specific Crusader vehicle),

$T(t_j)$ = task t_j (in this case, the route planning task),

n = the number of merit factors associated with execution of task t_j by vehicle v_i,

f_n = the appropriate satisfaction function for merit factor *n* (and task t_j on Crusader vehicles),

x = the raw data (for example, average time in which Crusader vehicle v_i executed task t_j),

and

w_n = a weight assigned by the experimenter to merit factor *n*, subject to the following constraint: $\sum_n w_n = 1$.

The weights, w_n, provide the experimenter with a useful capability to tailor the FOM for a given task and vehicle type to reflect the relative importance of its composite merit factors. For example, military domain experts may judge the speed with which a Crusader howitzer plans its routes to be generally a more significant consideration, in evaluating its execution of the task, than the "directness" of the routes that it plans.

Although computation of a set of FOMs (for Crusader vehicles in an exercise, executing a number of different tasks) is an important first step, the principal utility of the Value Calculus is realized when it *combines* these results to answer questions such as the following:

For "move" *missions*, in the uniform *battlefield* scenario, is the *applied value* of adding an IDSS with *route-planning* capability greater for *platoons* of Crusader *howitzers* than for platoons of Crusader *resupply vehicles*?

To deliver quantitative answers for such questions, the Value Calculus must support computations over *units* composed *of multiple vehicles*, and *operations* comprised of *multiple tasks*. For this purpose, the following general equation is applied, producing real-valued measures of *applied value*, V, on the range [0,1]:

(2) $V(\text{UNIT}, \text{OPERATION}) = 1/(m+n) \sum_m \sum_n w_{m+n} \text{FOM}(V(v_n), T(t_m))$

where

m = number of tasks, t_m, that comprise OPERATION,

n = number of vehicles, v_n, that comprise UNIT,

$\text{FOM}(V(v_n), T(t_m))$ is the primitive real-valued Figure of Merit, on the range [0,1], previously computed according to equation (1) for vehicle v_n executing task t_m

and

w_{m+n} = weights assigned by the experimenter to reflect relative impact of particular unit operations upon outcome of the simulated battle, subject to the following constraint: $\sum_{m+n} w_{m+n} = 1$.

It is worthy of notice that equation (2), in application to the suggested experimental question, would determine an average Figure of Merit for all Crusader howitzers or resupply vehicles in a platoon and for all tasks required to execute the "move" mission. This allows the applied value, V, to capture possible *secondary* effects that may result--in a realistic battlefield context--from the IDSS assistance with route planning. For example, it is conceivable that more efficiently planned routes may enhance performance of the route *following* task that is also a part of the "move" mission. Recognizing that interdependencies of this sort are by no means uncommon in the real operations of complex organizations (including hospitals and business corporations), we propose an evaluative methodology that does not "throw away" pertinent information about secondary effects.

Our provision of weights for the terms in equations (1) and (2) also permits use of information about certain *causal* relations that affect assessment of applied value. In the case of simulated military exercises, for example, After Action Review (AAR) typically reveals critical operations by units (or even key tasks performed by individual vehicles) that trigger chains of cause and effect with important consequences for the outcome of the battle. Relatively more weight must be assigned to such "key" events if we expect to assess accurately the applied value of proposed IDSSs. Used in conjunction with appropriate software resources for capturing data about such events from the simulation network, our Value Calculus provides a simple mechanism for including them in computation of applied value. This provision, moreover, should be equally useful for analyzing data from business or medical simulations. Indeed, the operation of nearly any complex system can be expected to involve "critical" events in which even relatively small advantages from IDSSs may play an important causal role in determining overall success or failure of the system.

7. Conclusion

Published methodologies for evaluating IDSS technology have tended to neglect the important need to measure its organizational impacts--i.e., to quantify its *applied* value. We have presented the outline of an approach that directly addresses this need. The three major components of our solution recommend use of the DIS simulation infrastructure, exploitation of existing software resources that support "intelligent objects" and application of a simple Value Calculus for data analysis.

8. References

Ceranowicz, Andy; Ladd, Carol; Smith, Joshua; and Vrablik, Robert. ModSAF SOFTWARE ARCHITECTURE DESIGN AND OVERVIEW DOCUMENT. Orlando, FL: Loral Systems Company, 1993.

Cohen, Paul R., and Howe, Adele E. "How Evaluation Guides AI Research." AI Magazine 9 (Winter 1988): 35-42.

Harrison, Patrick R. and Harrison, Ann P. "Validating an Embedded Intelligent Sensor Control System." IEEE Expert (June 1994): 49-53.

Heathfield, H. A., and Wyatt, J. "Philosophies for the Design and Development of Clinical Decision-Support Systems." Methods of Information in Medicine 32 (1993): 1-8.

Kirani, Shekhar; Zualkernan, I. A.; and Tsai, W. T. "Comparative Evaluation of Expert System Testing Methods." Technical Report, University of Minnesota, Computer Science Department, April 28, 1992.

McGee, J.; Metzler, T.; and Paesano, S. *Pre-Planned Product Improvement Study: Advanced Command & Control Evaluation.* Groton, CT: General Dynamics - Electric Boat Division, June 28, 1991.

Mohn, Howard Lee. "Implementation of a Tactical Mission Planner for Command and Control of Computer Generated Forces in ModSAF." Monterey, CA: Naval Postgraduate School, Master's Thesis, September 1994.

Preliminary Operational Concept for Advanced Field Artillery System (AFAS) and Future Armored Resupply Vehicle (FARV). United States Army Field Artillery School, (DRAFT) 3 March, 1994.

The DIS Vision: A Map to the Future of Distributed Simulation. Version 1. Prepared by DIS Steering Committee. Orlando, FL: Institute for Simulation & Training, May, 1994.

Wyatt, J., and Spiegelhalter, D. "Evaluating medical expert systems: what to test and how?" Medical Informatics 15 (1990): 205-217.

9. Authors' Biographies

Theodore Metzler is a Systems Engineer at LB&M Associates, Inc. Mr. Metzler has an M.S. degree in Computer and Communication Sciences and a Ph.D. in Philosophy. His research interests are in the areas of Hybrid Artificial Intelligence and Artificial Neural Networks.

Joseph Kelly is a Systems Analyst at LB&M Associates, Inc. His educational background is in General Business, Computer Systems Analysis and Material Acquisition Management. His research interests are in the areas of Logistics, Human Factors, Graphical User Interfaces, Artificial Intelligence/Expert Systems.

Supporting Materiel R&D Using Linked Engineering, Constructive, and Virtual Modeling and Simulation Tools

John A. O'Keefe IV
US Army Natick Research, Development and Engineering Center
Natick, MA 01760-5015
jokeefe@natick-emh2.army.mil

Robert McIntyre
Simulation Technologies, Inc.
111 W. First Street, Suite 748
Dayton, OH 45402
rmcintyr@natick-emh2.army.mil

1. Abstract

In a world of ever decreasing funding and accelerated development programs designers of protective equipment can no longer depend solely on the traditional methods of trial and error. Recently, developers of individual ballistic protective clothing have been required to reexamine such issues as non-homogeneous construction, increased body coverage, new materials, and rapidly evolving threats.

Traditional engineering models used in the design and evaluation of individual ballistic protective clothing compare the maximum possible serious and lethal wounds that would be expected for a given design worn by a standing man exposed to a fragmentation munition. The models are useful in making system comparisons for potential injuries. They do not provide a result that can be easily translated or related to an operational setting.

Wargames such as Janus and CASTFOREM provide a means to model such operationally relevant issues as loss exchange ratios and time to engagement in a controlled operational environment. They do not, however, provide a means to examine human factors issues such as fatigue, heat stress, the effects of environment on human movement speed, or a means to develop probability of detection, probability of hit, probability of kill from engineering descriptions of proposed clothing and equipment.

Virtual simulations such as SIMNET, SAFDI, CCTT, and MODSAF provide an opportunity to insert the human into the loop minimizing much of the traditional administrative overhead associated with traditional wargames. They also provide extremely vivid visualization tools in the form of computer image generators (CIGs) and Stealth Workstations that, if properly used, allow decision makers to view proposed equipment operating in a realistic tactical setting before it is even prototyped.

During the last year, the 21st Century Land Warrior Integrated Technology Program (21CLW ITP) and the Generation II Soldier Advanced Technology Demonstration (GEN II ATD) have been examining concepts for inclusion in their technology demonstrations. One of the issues that both 21CLW ITP and GEN II ATD are seeking to illustrate is how to use technology to make dismounted individuals more effective while increasing their survivability. In preparation for the concept development phase of GEN II a large number of material combination and area of coverages had to be rapidly examined to identify three ballistic fragmentation protective ensembles that weighted 13, 14, and 15 lbs. while minimizing the expected levels of casualties, heat stress and fatigue. The required analysis was accomplished using a linkage of engineering casualty assessment models, linear programming tools, performance models, and DIS simulations.

The analysis identified the three proposed systems, studied their impacts on heat stress fatigue and survival and provided visualization of the effects of the three systems plus two existing baseline systems over a period of six weeks. The resulting systems have become the basis for an Body Armor Advanced Warfighting Experiment (AWE). The analysis has been extended to include the effects of individual small arms.

The analysis has won the FY94 Army Materiel Command Group Systems Analysis Award and the performance model, the Integrated Unit Simulation System (IUSS), used in the analysis was used during the 16th Interservice/Industry Training Systems and Education Conference (I/ITSEC) to provide individual computer generated dismounted infantry.

2. Introduction

US Army research and development programs are being required to use a full spectrum of modeling and simulation (M&S) tools to support design and investment decisions. These M&S tools include engineering models, casualty assessment models, performance simulations, wargame simulations and Distributed Interactive Simulations (DIS). While many of these M&S tools have existed for quite some time, they have not been previously linked in coherent support of a materiel development program. Recent US Army guidance from the Assistant of the Army for Research Development and Acquisition requires that all Army Advanced Technology Demonstrations (ATDs) must address the use of DIS in published Simulation Support Plans (SSPs). The use of DIS to support materiel development decisions is still very much in its infancy. A prototype analysis using a full spectrum of M&S tools including DIS was accomplished for the 21st Century Land Warrior Integrated Technology Program (21CLW ITP) during March-April 1994. This prototype analysis examined the potential of using M&S tools to optimize individual ballistic fragmentation protective ensembles. The M&S paradigm developed for this analysis has become the cornerstone of the 21CLW ITP SSP and the SSP for a number of other ATDs.

3. Background

Basic and applied Department of Army and Department of Defense research and development programs are today required to shorten the time to develop solutions to battlefield deficiencies. At the same time these programs have significantly reduced budgets and constantly increasing program oversight and review requirements. Modeling and simulation have the potential to aid these materiel development programs meet the requirement to shorten the research and development schedule while providing the data necessary to answer the requirements of the many program review and oversight requirements.

Recently the US Army Soldier Systems Command - Natick Research, Development and Engineering Center (Natick) used a suite of modeling and simulation tools to address the needs of Soldier System materiel developers. Over a period of six weeks, modeling and simulation tools were used to examine over 75 different body armor configurations to identify an optimal system that could be constructed for user testing. The optimal system needed to minimize the serious and lethal wounds from bursting munitions, minimize the effect on heat stress and fatigue, and not adversely affect the ability of the individual and small unit to accomplish a military mission across a wide range of environmental conditions. In addition, data was required to support the defense of the contribution of the proposed optimal armor system to individual survivability and mission accomplishment.

4. Approach

The required analysis was accomplished by linking widely accepted engineering level ballistic casualty assessment models with a human and small unit performance analytic simulation. The human and small unit performance model was operated in a Distributed Interactive Simulation (DIS) synthetic environment to provide visualization and a larger force than normally possible in a single model. The human and small unit performance analytic simulation generated aggregate data inputs for use in constructive wargame simulations. Each set of variables were executed repeatedly until sufficient data had been produced to support the statistical definition of the distribution of the results.

The approach used during this analysis was to:

- Model body armor alternatives (by body zone and by material)
- Select the four most promising alternatives based upon the modeling
- Model the four selected alternatives to produce lethal areas
- Calculate the system weights for each of the selected alternatives
- Simulate the tactical mission using the Integrated Unit Simulation System (IUSS)
- Develop inputs for Janus from the results of the IUSS simulations

The models and simulations that were used in this analysis were: the ballistic casualty reduction model CASRED and the Integrated Unit Simulation System (IUSS) Version 1.0. In addition, a linear program was developed to select a manageable number of proposed armor configurations from the large number modeled in CASRED for simulation in IUSS. Figure 1 illustrates the analytic flow used in this study.

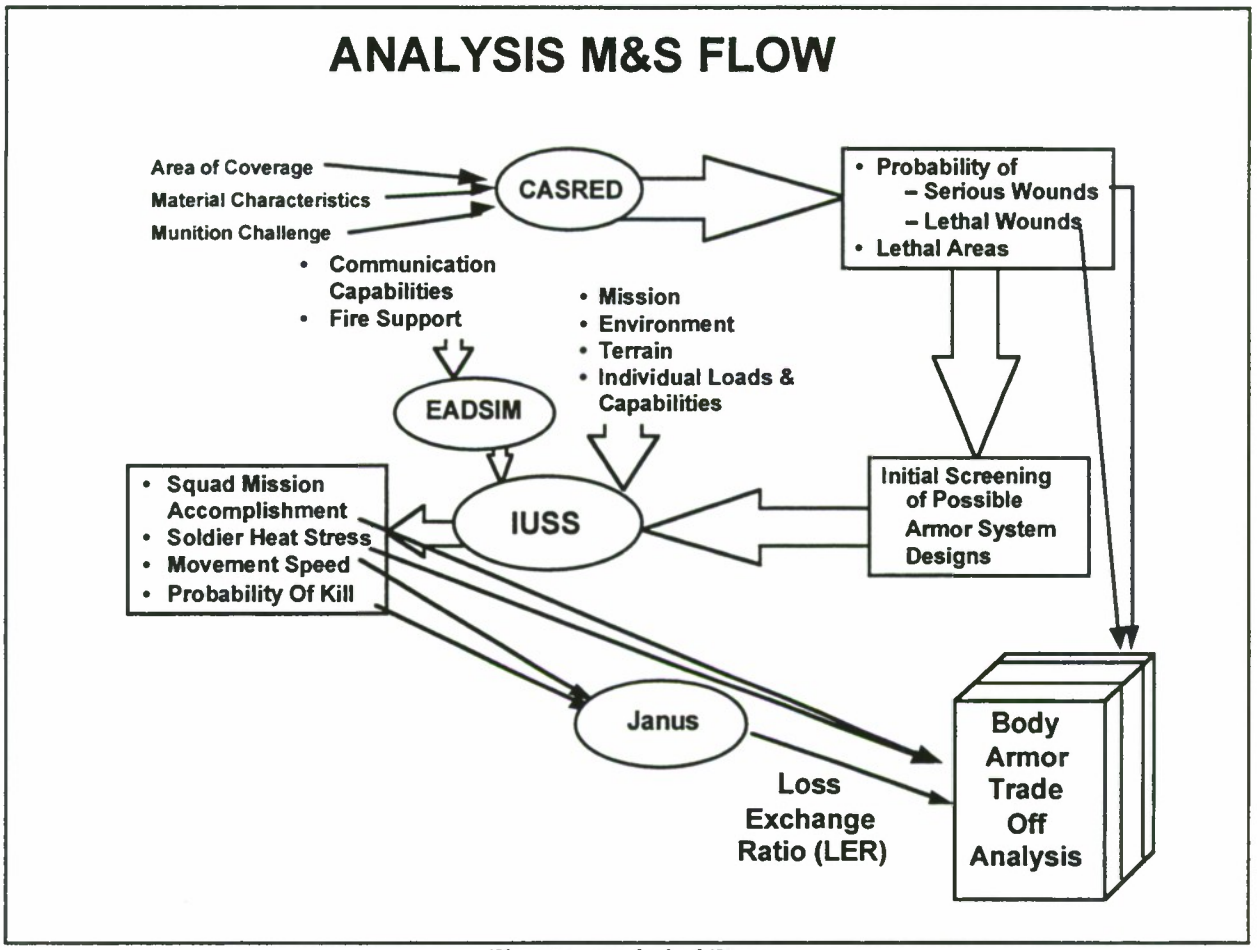


Figure 1. Analytical Flow

5. Discussion

Development of individual body armor has traditionally required extensive testing of materials followed by the building and testing of prototype systems. The testing of potential materials for use in proposed systems has required detailed testing using small panels of the material which are impacted with a representative set of ballistic fragment simulation projectiles. This testing generally costs between $25,000 and $75,000 for each proposed material and can require up to six months to accomplish. The exact cost and time of the testing is related to the availability of material, Government test facilities and the required levels of detail for the characterization of the ballistic protective properties of the material.

Following the ballistic panel testing, promising materials are fabricated into protective ensembles that undergo human use testing and further ballistic testing. Each of these prototype armor ensembles can cost as much as $5,000 and require six weeks to build. Engineering modeling using the CASRED, HELMETRAN, and ARMORTRAN ballistic casualty assessment models must be performed to provid a basis for comparison to existing protective equipment prior to type classification of a proposed body armor item.[1] This evaluation -prototype - evaluation & modeling cycle can be very time consuming and expensive, resulting in long periods between type classification of new protective ensembles for the soldier in the field.

The evolving worldwide political environment and US foreign policy, plus the need to rapidly insert new technologies into the field necessitates a reevaluation of how Soldier System equipment is developed and fielded. The Land Warrior (LW), 21st Century Land Warrior (21 CLW), and Generation II Soldier (GEN II) programs have further emphasized the need to

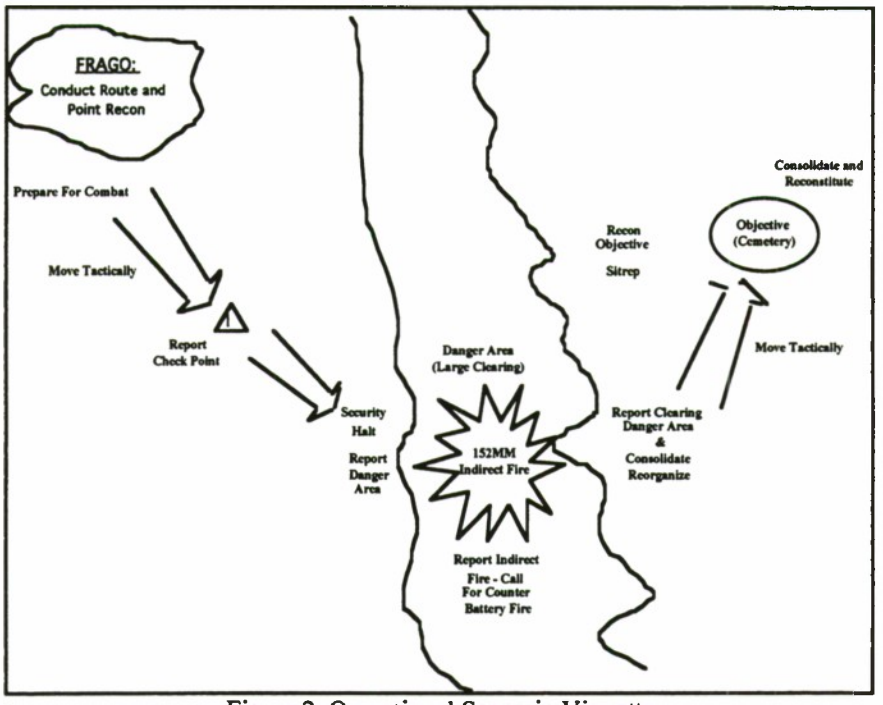


Figure 2 Operational Scenario Vignette

shorten the traditional Soldier System research and development cycle. Therefore, during March and April 1994, a consortium lead by the Modeling and Simulation Branch, Advanced System Concepts Directorate, Natick undertook the task of developing a new paradigm to shorten the time required to design Soldier System equipment using linked engineering, performance, DIS, and constructive models. This effort[2] used as input to the CASRED model the following information:

- Three armor materials,
- Five possible percentages of coverage for five zones of the body
- Two helmet collar combinations
- Seven threat munitions

CASRED is designed to calculate the serious and lethal wounds for each body region that statistically would occur for a given munition/protective system combination. These body region results are then summed for the entire body. The results for both the individual zones and the total body are calculated by the model. During the *Analysis of Ballistic Protection Concepts Quantifying Operational Impacts and Design Criteria*, the individual body zone results for each possible armor material/munition combination and the associated material weights were input to a linear program. This linear program identified the armor system configuration for any given system weight which provided the lowest possible expected lethal wounds while minimizing the expect serious wounds. Optimized ballistic protective systems were developed for system weights between 6 and 25 pounds. In addition, a set of design criteria based upon the results of the casualty reduction modeling and the linear program optimization were developed.

Based upon user input, three optimal systems were selected for simulation in an operational setting using IUSS and Distributed Interactive Simulation (DIS) tools. These systems were a 13 lbs., 14 lbs. and 15 lbs. armor system. In addition, the current Personal Armor System Ground Troops (PASGT) vest and helmet, plus systems composed of 100% coverage of all five body regions with each of the three armor materials were simulated in the same operational setting. Figure 2. graphically depicts the operational scenario.

For each pairing of alternative system, threat bursting munition, munition circular error probability (CEP), temperature and humidity, 250 iterations of the IUSS simulation were executed. During the IUSS simulations environmental conditions ranging between 10°C to 30°C, 25% to 95% humidity, and either night or clear day sky were examined. The results of these iterations were then statistically analyzed to determine the mean, median, and mode for heat stress casualties, serious wounds, lethal wounds, mission completion time, and movement speeds. The simulation results were also subjected to T test and Student T analysis.

DIS tools were used to provide visualization and data logging of the simulation exercises. The virtual reality visualization was used to perform rapid sanity checking of the tactics, techniques and procedures (TTP) used for the simulated dismounted infantry in IUSS. The data logs were used to provide replay of the simulation exercises. These replays allowed operational users and analysts to visually review all aspects of the simulation exercises and rapidly identify those event which warranted further detailed investigation.

6. Limitations

The casualty assessment models used to develop the expected casualty data for IUSS, Janus, and CASTFOREM contain a number of limitations. These include that the soldiers are assumed to remain in a standing position during the entire artillery attack and they simplify their representation of the human body by dividing it in to six cylindrical parts to which was assigned a uniform statistical probability of serious or lethal wounds based upon a residual kinetic energy for the various presented fragments. The available computational capabilities necessitated these model simplifications when the casualty model was originally developed. Newer techniques are now available to support more detailed calculation of ballistic casualty for a dynamic array of body postures during a ballistic attack.

7. Study Results

The results of 250 IUSS simulation runs for each body armor alternative (PASGT Helmet, PASGT Vest plus Helmet, 13 lb. Optimized System, 14 lb. Optimized System, and 15 lb. Optimized System) showed that ballistic protection and soldier dispersion/tactical employment must be balanced.

While the CASRED results show the maximum possible lethal and serious wounds if soldiers were located at every possible location around the detonation of the munition, IUSS illustrates that less dynamic casualty results will be observed when soldiers are dispersed and maintain a minimum of 35 meters spacing.

The simulation results are influenced by the capabilities of the CASRED model. Currently, only casualties for standing soldiers are calculated. In a battlefield setting, soldiers will tend to initially go to ground while they identify where the attack is coming from. This would require the casualty calculations, first, to be calculated for standing and then prone soldiers.

The Distributed Interactive Simulation (DIS) version of IUSS was used in conjunction with a Photo Realistic DIS Stealth to illustrate the findings of the analysis. These tools allow the rapid visualization of the effects on movement and survivability that each of the simulated alternatives produced.

The following observations were made during the analysis:

- A 25% increase in protective equipment system weight is a 5% increase in the total combat load
- 5% increase does not create an operationally significant increase in the impact of total load on marching speed or mission completion times
- However, this increase may produce a significant increase in ballistic survivability

8. Future Plans

Since the initial analysis was completed additional analyses have been conducted to examine the effects of current and proposed body armor on expected small arms casualties. The technique has been expanded to simulate larger units of dismounted infantry using IUSS simulations linked using a DIS network and DIS communication protocols.

The operational mission has also been expanded to include a larger friendly force engaging an opposing infantry force. This scenario will be progressively expanded to include more and more of the total combined arms team.

Simulation of additional operational missions are planned. These planned missions include Military Operations in Built-up Areas (MOBA) and Operations Other Than War (OOTW).

9. Potential Uses

While conducting the *Analysis of Ballistic Protection Concepts Quantifying Operational Impacts and Design Criteria* the potential power of DIS visualization tools to help decision makers rapidly review the application of hypothetical equipment was demonstrated. This was accomplished by describing the proposed equipment's characteristics in IUSS and transmitting the individual activities of each soldier across a DIS network using Entity State Protocol Data Units (PDUs). These Entity State PDUs were then used to animate a "Photo Realistic" Stealth Workstation that displayed the activities of the soldiers. This linkage of extremely high fidelity

computer simulation of soldier performance and survivability with the stealth workstation provided a rapid visualization of proposed equipment, procedures and tactics helping to debug and understand the outcomes of the simulation exercises. The remainder of the synthetic environment could be populated using Modular Semi-Automated Forces (MODSAF), other computer generated forces and/or manned simulators.

Another area with rich potential is the use of data logger tapes from previous training and mission rehearsal simulation exercises to establish a scenario setting into which proposed technologies can be inserted. This would free the technologist from the potential pitfalls associated with the population of a realistic synthetic environment with systems and forces that are unfamiliar to them. Instead, the technologist would be free to concentrate on the simulation of their proposed technology, depending on the previously capture synthetic environment to describe the other forces and systems operating in the synthetic environment.

10. Conclusions

This analysis has shown how engineering and integrated performance models and simulations can be used to develop inputs usable for Combat Models such as Janus and CASTFOREM. It provides a methodology for optimization of material configurations which resulted in significant increases in protection over current equipment. The 15 lb. optimum system provides the maximum reduction of serious and lethal wounds across the fragmentation munitions with the least negative impact on mobility, heat stress, and mission accomplishment. The usefulness of DIS tools to support analytic efforts was demonstrated during this analysis.

The analysis also highlighted the need to improve engineering and CASRED methodologies to address the effects of: different soldier positions; vital organ protective strategies; small arms and flechettes. It also showed that the expected ballistic fragmentation casualties could be reduced by more than 30% when a structured system approach was used.

11. References

[1]O'Keefe, John A. IV, "Casualty Reduction Modeling" , Natick TR/89-00 , Natick, MA,

[2]O'Keefe, John A. IV, et. al., "Analysis of Ballistic Protection Concepts Quantifying Operational Impacts and Design Criteria", Natick, MA, 26 March 1994

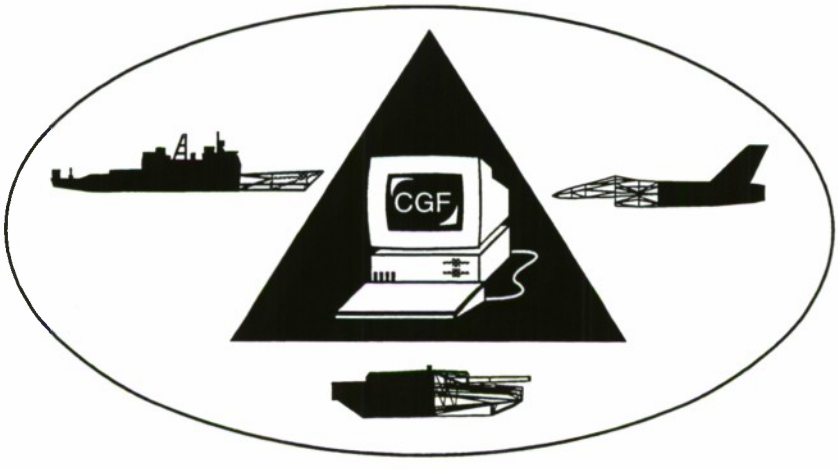
12. Author's Biographies

John A. O'Keefe IV, a graduate of Norwich University (BA 1975), American Technological University (MA 1981), the U.S. Army Infantry Officer Basic Course, the U.S. Army Infantry Mortar Platoon Course, The U.S. Army Ordnance Officer Advanced Course, the U.S. Army Combined Arms Service Staff School, the U.S. Army Inspector General Course, the U.S. Army Command and General Staff Course, and the U.S. Army Materiel Acquisition Management Course, and a disabled Regular Army Major, is a senior operations research analyst with the Advanced Concepts Division, Advanced Systems Concepts Directorate, U.S. Army Natick Research, Development and Engineering Center. He is the project officer in charge of the development of the Integrated Unit Simulation System (IUSS), Dismounted Infantry Support System (DISS), and the Soldier Protective Equipment Computer Aided Design (SPE CAD) system and is the Chairman of the Modeling Working Group of the U.S. Army Soldier System Technology Base Executive Steering Committee. He has been active in the application of modeling and simulation technologies to support materiel development since 1988.

Robert McIntyre, III, Vice-President, Operations for Simulation Technologies, Inc. He is the Simulation Technologies, Inc. Program Manager for IUSS. Mr. McIntyre is a graduate of the University of Alabama, Birmingham, where he studied Bio-Physics and Management Science.

Session 8b: Terrain Modeling II

Stanzione, TASC
Watkins, SAIC


CGF

Integrated Computer Generated Forces Terrain Database

Thomas Stanzione
Forrest Chamberlain

TASC
55 Walkers Brook Drive
Reading, MA 01867
tstanzione@tasc.com
flchamberlain@tasc.com

Dr. Alan Evans
Cedric Buettner

SAIC
486 Totten Pond Rd.
Waltham, MA 02154
aevans@bos.saic.com
buettner@bos.saic.com

1. Abstract

The Integrated Computer Generated Forces Terrain Database (ICTDB) project, being developed jointly by TASC and SAIC, is part of the ARPA/TEC Advanced Distributed Simulation Synthetic Environments program. The main goal of this project is to develop an integrated terrain representation that will satisfy the Computer Generated Forces (CGF) environmental reasoning requirements for ARPA's Synthetic Theater of War (STOW) program. This representation will address many of the shortcomings of current CGF terrain databases, as well as include a richer set of terrain features and attributes for advanced terrain reasoning. As with previous CGF terrain databases, this database will contain terrain features as spatially organized objects. However, irregular terrain grids, multiple elevation features such as tunnels and bridges, and feature aggregation for higher echelon terrain reasoning will all be supported. This representation will allow for real time updates in order to handle dynamic terrain and weather effects. The representation will provide CGF terrain support for the other Synthetic Environment programs. A few terrain databases will be generated in this representation and demonstrated using ModSAF.

2. Requirements Analysis

The first phase of this project, which was completed in March, consisted of a requirements analysis, data source investigation, and preliminary design. In the second phase, an incremental development is underway to provide the components of the representation to support the more critical STOW requirements. A series of engineering demonstrations is scheduled at TEC through the summer and fall of 1995 to test these changes within the ModSAF environment. This paper focuses on the results of the work performed in the first phase of this program.

The requirements analysis task started with a literature review of current and some future computer generated forces, command forces, and constructive simulations in order to determine the terrain content, representation, analysis, and reasoning requirements of a wide variety of these systems. From this review, we developed a Requirements Survey Form that was used as a basis for our program interviews. We chose a number of specific programs to interview as part of this requirements analysis in order represent a broad spectrum of CGF terrain users and programs:

- ModSAF
- CCTT
- CFOR
- STOW
- War Breaker / JPSD
- Eagle

The STOW program requirements analysis was a very important part of this task. There are many terrain implications in the STOW requirements that this project needs to address. STOW will be a Joint Services training program, which implies that a variable resolution terrain representation is needed to accommodate individual combatants, ground entities, and high flying aircraft. This also implies that an ocean and littoral region representation is needed. STOW requires the integration of virtual, live, and constructive simulations, which implies that the terrain databases used in the simulations need to be correlated to the real world as well as each other. STOW terrain databases will be much larger than previous near-ground databases, on the order of hundreds of a kilometer on a side, so a global coordinate system that allows for compact terrain representations is necessary. STOW will also require the simulation of tens of thousands of entities, so all

terrain services must be efficient. STOW requires that the simulation environment must be dynamic, which requires that terrain databases must be able to be updated in real-time for battle damage, combat engineering effects, and weather effects.

We also performed a review of terrain analysis procedures in order to insure the terrain representation will provide the appropriate terrain data. These analyses included mobility and trafficability models, cover and concealment determinations, and identification of landing zones. A draft Requirements Analysis document (Stanzione & Evans 1994) and draft Application Programmer's Interface document (Evans et. al. 1994) were generated and distributed to a number of organizations for review:

- ARPA (PM Synthetic Environments, PM Synthetic Forces)
- TEC
- STRICOM
- Loral
- Mitre
- MIT Lincoln Labs
- NPS
- SRS
- TRAC, Ft. Leavenworth

From the results of these analyses and reviews, a common set of requirements for the ICTDB representation was determined and is shown in Table 1. The requirements were grouped into two categories in order to distinguish those requirements that are already provided by current CGF terrain databases and required to maintain current functionality, and those requirements that are currently not supported to the level necessary for STOW.

A data source definition task was also performed in phase one of this project. It focused on populating the ICTDB representation from integrated terrain datasets consisting of Triangulated Irregular Networks (TINs) for elevation data and feature data from operational data sources, such as Interim Terrain Data (ITD). The combined capabilities of ARC/INFO and the S1000 toolkit were examined as the primary terrain database generation tools, with particular focus given to the S1000 Application Programmer's Interface. A Data Source Definition document (Buettner, et. al. 1995) was generated as part of this task.

Table 1:
Requirements for ICTDB

Basic Requirements	Advanced Features
ANSI C (with ADA interface)	Global coordinates w/ local cartesian
Vehicle placement	Storage of integrated TINed surfaces
Elevation lookup	Multiple LOD elevation data
Soil type and slope queries	Dynamic updates
Negative elevation values	Non-homogenous aggregate features
Multiple elevations at location	Expandable features and attributes
Line-of-sight and area intervisibility	USCS soil types and other mobility attributes
Spatially organized	Mobility corridor networks
Road and river networks	Building interiors
General feature type queries	Precipitation and temperature effects
Contour line generation	Sea state and sea floor representation
	Expandability of database during generation
	Partial database loading
	Check pointing terrain
	Multiple terrain views

3. Representation Design

The ICTDB representation design task consisted of developing an overall design for the integrated terrain information, as well as designs for the individual components. The ICTDB representation consists of three integrated components: a local terrain component for elevation and trafficability information, a global terrain component for terrain reasoning information, and a feature component for specific feature and attribute information. A global coordinate system designed to support near-ground exercises on any scale was also defined. Figure 1 shows an overview of this design.

3.1 Coordinate System

As part of the ICTDB design phase, a white paper on the use of coordinate systems by CGF simulations has been written (Evans & Stanzione, 1995). This paper spells out the reasons for using a *global coordinate system* which differs somewhat from what is currently in use in most CGF applications.

Simulations participating in a DIS exercise are constrained to use a common *public* representation of coordinates in the virtual world, as specified by the DIS protocol. This public representation is, of course, Geocentric Cartesian Coordinates (GCC) with coordinate values represented using 64-bit double precision vectors. A good summary of the rationale behind using GCC for DIS can be found in the BBN white paper (Burchfiel & Smyth, 1990). GCC is based on a right-handed Cartesian system with its origin at the Earth's center, the X-axis passing through the Equator at the Prime Meridian, the Y-axis passing through the Equator at 90 degrees east, and the Z-axis passing through the North Pole. This representation has obvious advantages. GCC is a *real-world* system. Put simply, straight lines in GCC represent straight lines in the real world, unlike systems which involve coordinate projections. Furthermore, GCC is inherently extensible to exercise regions of arbitrary size. The GCC representation is however, very unwieldy for internal use by a simulation application, since the magnitude of the numbers required to represent vectors near the surface of the Earth is quite large. Furthermore, the interpretation of GCC coordinates at points on or near the Earth's surface is not intuitive, since GCC is not a local frame of reference. This is clearly an obstacle to the developers of code implementing behaviors, platform kinematics, sensors, weapons systems, and so on within a CGF application.

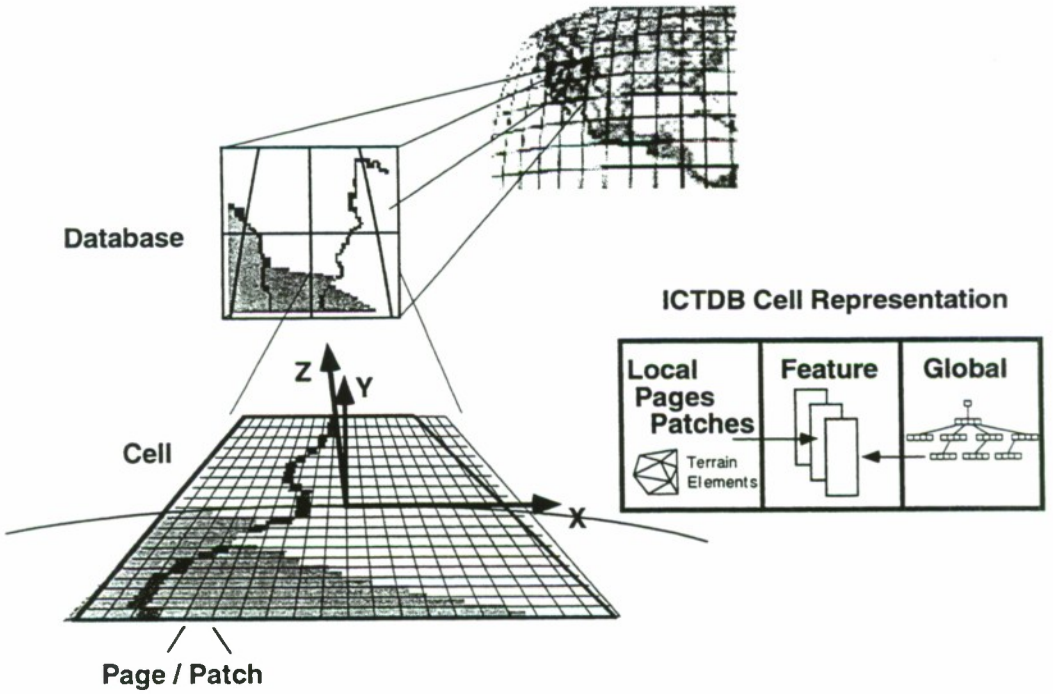


Figure 1: ICTDB Overall Design

Current CGF systems which simulate ground entities usually use some form of local Cartesian coordinate system with position specified as an offset vector from a fixed origin, typically the southwest corner of the database. The terrain data are derived from a representation that is a planar projection of some form, most often a UTM representation. Indeed, prior to the adoption of DIS as an IEEE standard, the SIMNET protocol specified a *public* representation on the network of coordinates as X and Y offsets in meters from an origin at the southwest corner of the database in a suitable UTM grid zone.

There are two fundamental problems with coordinate systems based on UTM representations. First is the difficulty of scaling to large exercise areas. In order to participate in an exercise spanning multiple UTM grid zones, a CGF application would be forced to manage transformations from one local (projected) frame of reference to another. These transformations would need to take place quickly and be transparent to the behavioral and platform models. A second problem is encountered in the inherent simulation anomalies that result from the fact that "straight" lines parameterized in such systems actually represent curves in the real world, owing to the transformations employed in planar projection. In past exercises spanning UTM grid zones, database coordinates have simply been extended beyond grid zones specifically to avoid transformation difficulties. This leads to even greater projection anomalies. Of course, DIS requires that coordinates be represented in GCC when being transmitted on the network. Therefore, efficiency concerns suggest that future CGF systems will represent locations in the virtual world using an internal representation easily converted to GCC. Because developers tend naturally to write code which is based on linear parametric equations, a Cartesian coordinate representation is also suggested to avoid the anomalies due to projection.

The following are the fundamental simulation requirements which affect the choice of coordinate representation in a CGF system:

- Scaleability - As simulations grow larger in scale, it is crucial that an internal representation support arbitrarily large exercise areas, perhaps even the entire surface of the Earth.

- Compactness - Storage must make efficient use of space. This requirement is, of course, closely related to the scaleability requirement.

- Faithfulness - The coordinate representation should be free of anomalies such as curvature effects.

- Ease of translation to and from GCC - For efficiency reasons, an internal coordinate representation should support fast conversion both to and from GCC, since this translation must occur for every location vector which is read from or written to the DIS network.

- Naturality - The coordinates values returned to code simulating platforms and command elements must be *natural* in the sense that they must have an intuitive relationship to the real world for the benefit of developers of this code.

The ICTDB project has chosen a coordinate representation which addresses all of the requirements listed above. Based on our survey of CGF systems, we believe that ICTDB is building the first simulation subsystem for storing and accessing digital topographic data that, in fact, meets all five requirements identified above: scaleability, compactness, faithfulness, ease of translation to and from GCC, and naturality.

In elaborating a design for ICTDB, we have chosen a hierarchical database organization, which in principle can be scaled to accommodate exercises of arbitrary size. At the highest level, the surface of the Earth is subdivided into *cells*, one degree in latitude by one degree in longitude. A cell is thus approximately 100 kilometers by 100 kilometers square. This terminology is borrowed from the War Breaker World Reference Model (Brockway & Weiblen, 1994). Within a cell, ICTDB data are further organized into *pages*, which are in turn subdivided into *patches*. Patch size can vary from database to database, depending on feature density. The page is really a logical unit used to fetch and store data. The patch is the fundamental unit of organization for both features and TIN data. While storage limitations will certainly be a factor in building large databases, ICTDB has met the scaleability requirement by providing the ability to support large databases that can span multiple cells.

Within each cell, a local Cartesian frame of reference is defined. The origin is at the center of the cell, with

the X axis pointing east, the Y axis pointing north, so the X-Y plane is tangent to the Earth's surface at the origin of the local frame of reference, and the Z axis is an outward normal vector to the Earth's surface. The requirement of naturality is met by using Cartesian frames locally. It must be noted however, that the ICTDB cell coordinates do not furnish a Z vector which is normal to the Earth's surface, except at the origin of each cell frame. There is a deflection from the vertical which increases toward the edge of each cell, up to a maximum value of about 0.7 degrees. While not significant for ground vehicles, it will be necessary for the ICTDB API to provide an inward normal, if requested, at a given point (X,Y) in each cell frame. This will be especially important for long range ballistics calculations.

In order to convert GCC coordinates to the ICTDB representation, the cell number is first calculated by a geodetic transformation. This yields an pair of indices which reference the cell data. Each cell has an associated offset vector and 3D rotation matrix. Vectors in the cartesian coordinate frame in the cell are simply GCC vectors, with the offset vector subtracted, that are then multiplied by the cell rotation matrix. This means that the transformation from GCC to cell coordinates is linear. Of course, the inverse transformation from cell coordinates to GCC is also linear. This meets the fourth requirement that coordinates be easily transformed to and from GCC. Furthermore, since the transformations are linear, the ICTDB global coordinate system meets the faithfulness requirement, unlike projected coordinate systems.

We note that a similar local Cartesian coordinate system has been in use for some time in some airborne radar systems. An analysis of some of the tolerances involved in the paper (Gadeken, 1976) suggests a larger cell size, say five degrees by five degrees, may be appropriate. The smaller cell size in ICTDB is actually tied to the compactness requirement. We have elected to store elevation values for TIN data in a compact fixed point format similar to the ModSAF CTDB implementation (Smith, 1994). Use of the larger cells would require floating point elevation data, hence would use more storage. In addition, we estimate that the overall storage requirements for such larger cells would be on the order of magnitude of a hundred megabytes instead of four or five megabytes, which is unacceptable for system performance. These storage considerations, together with the much larger gravitational deflection in large cells, led to our choice of a relatively small cell size.

Most calls to the ICTDB API will require that a complete set of coordinates in this global reference system be passed. That means that the cell, or a pointer to the cell, along with the X and Y in the cell cartesian coordinate frame are needed to specify a location. We considered defining a *current cell* with all calls referencing the current cell by default. While this would make porting ICTDB into existing applications a little easier, it would blur the interface and introduce ambiguity.

Since the ICTDB API defines coordinates as a type which includes not just the X,Y and Z in the cell frame, but also a reference to the cell itself, ICTDB will need to support operations on these coordinates, that is a vector algebra for cell coordinates. For example, an application often needs to calculate an offset vector from a platform location A to another entity location B, for an intervisibility calculation, say. If A and B are in different cells, the application cannot simply subtract vectors. The ICTDB library will have to provide these vector operators, to assist in cell transitions.

3.2 Local Representation

The ICTDB local representation has been designed to contain all elevation data and physical feature data. The physical feature data are divided into two parts: geometric data and attribute data. Geometric feature data are maintained within the local representation, while the attribute data, not required for time critical functions, is accessed by referencing the feature component of the representation. The rational for maintaining geometric data locally is to minimize data access time, thereby minimizing the terrain representations effect on time critical routines (i.e., elevation lookup and intervisibility).

The fundamental assumptions behind ICTDB's local representation is that terrain surfaces will be primarily integrated TINs. The representation will maintain the minimum set of data required for terrain services, so data access is efficient for time critical routines (elevation lookup and intervisibility). Although we expect most future databases to be generated from integrated TINs, our design will not preclude the use of gridded terrain. We are addressing the merits of two approaches. If the database cell is completely gridded, then we may use the ModSAF representation to

handle the cell. If the database is partially TINned, then the non-TINned grid posts could be converted into two triangles and accessed using the point location algorithm mentioned below. By changing grids into TINs, there would be a slight increase in database size (duplication of stored vertices) and a small performance penalty (extra level of indirection). For the TINed portion of a database, the elevation data will be stored as terrain elements in the local representation. A terrain element is either a triangle (3 vertices) or a polygon (4 vertices). Each terrain element will contain all vertices, edges (implicit or explicit), soil information, and adjacency/topological information (neighbor to my edge). The representation also includes new data structures called virtual grids. These virtual grids enable a direct mapping from a point on the database to the most probable terrain element. By adding adjacency information, in addition to the virtual grid, neighboring terrain elements may be interrogated if the most probable terrain element does not contain the database point. Also, the intervisibility engine traverses through terrain elements, only looking at those elements that have a direct effect on intervisibility. In order to convert TINs to our prototype representation we used ModSAF's algorithms that produce the terrain element edges and vertices in patch relative coordinates. The data were augmented with adjacency information and then stored into the final format. The new representation requires some extra storage, but the storage cost is offset by the recycling of most of the grid post data. However, we do plan to keep the features present mask capability of the grid posts. Our current data show an increase in database size by 25-30 percent, but efforts are being made to decrease the size without impacting performance.

Currently the design and profiling effort has focused on improving the performance of two terrain services routines (intervisibility and elevation lookup) in a TINned database. To improve performance substantially, we need direct access to the terrain element which corresponds to a given position on the database. The point location algorithm we implemented was derived from an algorithm developed for use during the TINing process (Scarlatos, 1993). The approach overlays a TINned region (i.e., patch) with a grid, which we call a virtual grid. The number of rows and columns in the virtual grid is based on the square root of the density of the terrain elements (triangle) rounded to the nearest integer. For example (see Figure 2.), if there were 10 terrain elements in a region then the virtual grid would be a 3x3 virtual grid. Next, one terrain element is mapped to a virtual grid, if the area of intersection between the terrain element and the virtual grid is a maximum of all terrain element intersections (see shaded areas within Figure 2.). This ensures that the point location algorithm will first interrogate the most probable terrain element associated with a virtual grid. If the point is not within the mapped terrain element, the adjacent terrain elements are interrogated using the topological information about neighbors accompanying each terrain element. If the point is not within the neighboring terrain elements, then their neighbors are interrogated. If the point is not within these neighbors, then all remaining terrain elements will be interrogated. A mask containing the index of all terrain elements interrogated will be maintained to prevent duplicate interrogation of a terrain element. We are investigating additional optimizations on this approach.

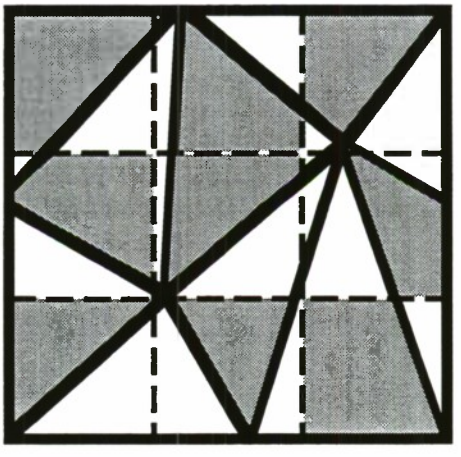


Figure 2: Virtual Grids

Current CGF terrain representations are highly optimized, and since our goal is to improve performance when using TINs, we needed to prototype and profile our local representation designs to get a feel for the potential storage and performance costs. We used ModSAF 1.3 libctdb as the timing and storage baseline. To facilitate the profiling effort the following steps were taken.

1. Reformatted a TINned CTDB database (local representation) into a prototype ICTDB format.

2. Generated a new terrain service algorithm that exploited the new representation.
3. Generated a profiling routine that incrementally stepped over a database, determined the average terrain element density over the interrogated region, interrogated a set of random points, and then returned the average time required to interrogate each point.
4. Ran profiling tests on the same machine (SGI Indy R4400 with 96 MBytes RAM) and collected data using either the ICTDB or CTDB terrain representation and the applicable terrain service routine.

We expect the time required for elevation lookup to be "approximately" constant for the average case, since we have a direct mapping from a position to a virtual grid which in turn maps to the most probable terrain element. The "approximately" arises if the point does not map directly to the terrain element, requiring that the adjacent triangles be interrogated. Further prototyping will address effects of variable and decoupled (feature and terrain) patch size, summary data (i.e., maximum or minimum elevation data), terrain compactness, and conversion of grids to terrain elements in partially TINned cells.

3.3 Global Representation

The ICTDB global representation stores references to every feature in the database. References are organized to support queries at varying levels of detail requesting lists of features found in a specified area. Levels of detail are designed to meet the needs of units of varying echelon.

Past terrain representations for computer generated forces, such as the ModSAF Quadtree database, have organized features by location to support rapid retrieval of features found in an area. However, these representations generally retrieve a list consisting of every feature in the given area, presenting more detail than is required by higher-order echelons. The goal of the ICTDB global representation is to provide the capability to filter these lists to generate only features of potential interest to echelons of varying size.

The primary distinguishing characteristic determining whether a given feature is of interest to a unit of a given echelon level is its size: larger units are less likely to be concemed about smaller features. In addition, real-world commanders of larger units are likely to mentally group many smaller features, such as individual buildings, into a few larger features, such as city blocks or towns. Thus, it is important to support multiple level of detail queries via two mechanisms: selecting features based on their size, and grouping features together to form new features. To support this second mechanism, we define an aggregate feature to be an abstraction representing a group of two or more features, which may themselves be aggregate features. We believe that the basic query supported by the global representation should be of the form:

> Return all features in area A larger than size S. If possible, do not return multiple references to any feature at multiple levels of aggregation.

Building on past work in the field, we chose a quadtree as the basic data structure for the global representation. This provides the desired spatial organization. As stated above, the novelty of our approach stems from our treatment of multiple levels of detail queries. In the ICTDB global representation, each level of the quadtree implements a distinct level of detail. Any given node should be able to rapidly generate a list of all features that overlap it and are at least as large as the node itself. If both a feature and an aggregate containing that feature meet these criteria, the aggregate should be removed from the list, since it can be accessed through its association with the feature. This could easily be accomplished by storing a list of all such features at each node. However, doing so would result in extensive replication of references to large features.

To avoid undue repetition of references, a scheme was developed whereby two classes of reference are distinguished: explicit references and implicit references. Explicit references are pointers to features contained in a node. Implicit references are a means for a node to include features explicitly referenced at another node, thus limiting replication. Ideally, references should be made according to the following two rules:

1. Each feature is explicitly referenced at the level of the quadtree whose nodes are closest to it in side dimension. At that level, each node overlapping the feature contains a pointer to the feature.

2. Each feature is implicitly referenced at each node, N, descended from those nodes at which the feature is explicitly referenced after meeting the following criteria:

a. N overlaps the feature.
b. If the feature is an aggregate feature, N does not overlap any component of the feature at or above N's level in the quadtree.

Figure 3 shows an example of this referencing scheme. In this example, the large square represents level 0 of the quadtree, its four sub-squares are at level 1, and the four smallest squares in the upper-right corner are at level 2. The dotted line is the outline of an aggregate feature whose components are the black circles. Due to their respective sizes, the aggregate feature will be explicitly referenced at level 0, and the components will be explicitly referenced at level 2. Because the components are not referenced until level 2, all four level 1 nodes will implicitly reference the aggregate feature. Finally, at level 2 those nodes that overlap one or more components will explicitly reference those components and not reference the aggregate at all, while those that do not overlap any components will implicitly reference the aggregate feature.

With this scheme, there remains some duplication of explicit feature references. However, since features are of roughly the same size as nodes, each feature is unlikely to be explicitly referenced more than four times. Note that a combination of rule 1 and rule 2a would be adequate if there were no aggregate features; rule 2b ensures that each feature is accessed at only one level of aggregation.

While implementing rule 1 is straightforward, exactly implementing rule 2 would require some form of flag for each implicit reference, again causing waste of storage. We have chosen instead to approximate rule 2 through two mechanisms with very modest storage requirements. First, each node of the quadtree maintains a bitmask specifying at which of its ancestors it implicitly references at least one feature. And second, the list of features explicitly referenced at each node is sorted based on the maximum depth in the quadtree at which they should be implicitly referenced below the node. Using these structures, each node generates, at run time, the list of features it should reference in the following manner:

- Reference all features explicitly pointed to at the node itself.
- For each ancestor, if the bit mask specifies that at least one feature should be implicitly referenced, iterate through all features that can be implicitly referenced down to at least the level of the node. Any such features that overlap the node should be added to the list.

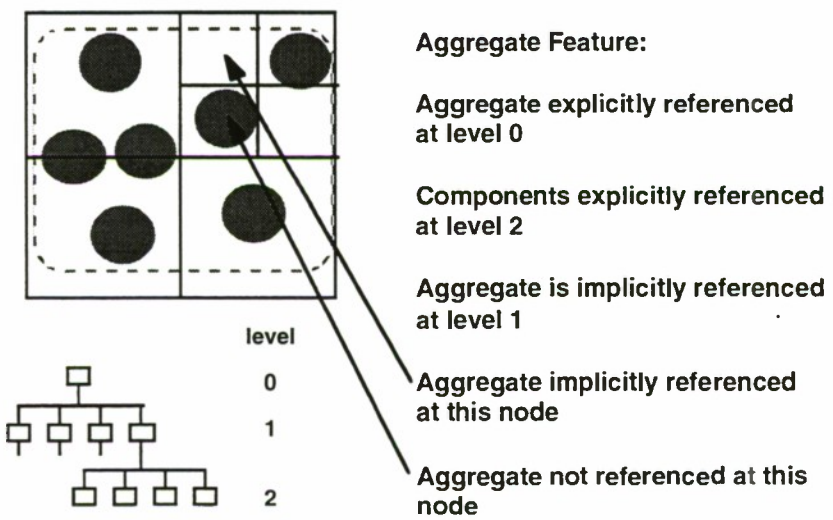


Figure 3: Aggregate Feature

This technique ensures that every node references every feature it should according to rules 1 and 2 above. In addition, nodes may reference some features at more than one level of aggregation. This is deemed an acceptable cost for the savings in space since it involves only the generation of redundant information.

Code implementing this representation has been written and tested. Performance analysis is still under way, but execution time is expected to be comparable to existing systems for simple queries, and superior for higher-echelon queries.

3.4 Other Design Considerations

Other factors were considered in the design process of the ICTDB representation. These include memory management, extensibility, topology, dynamic effects, and terrain views. Each of these topics is addressed in this section.

It is essential to limit the quantity of terrain data read by a CGF application to avoid consuming too much of the system's memory resources. This is balanced against the need to have all the data within an area of interest resident in memory to avoid slow disk accesses. In a simulation environment that supports dynamic terrain, this problem is compounded, since an application cannot be expected to predict with certainty which areas of a database will need to be modified as an exercise unfolds. Clearly, it is not possible to read an entire terrain database, since future databases may be many hundreds of kilometers on a side.

ICTDB has elected to allow the application to put a hard limit on the area of interest for terrain data. The API will provide a function *ictdb_set_aoi()* that specifies the database extents for an exercise. Outside this area of interest, updates to the database will be ignored. Initially, the area of interest will be specified using an application parameter file, or from the application's command interface, or both. Eventually, it would be desirable to build a GUI, perhaps with cartographic raster images, to provide the human user a convenient way to specify the area of interest.

Inside the geographic area of interest of an exercise, ICTDB will provide an additional function, say *ictdb_set_extents()*, that specifies the extents within which terrain data will actually be read when the exercise is started. All of the cell data necessary to cover the extents will be read at initialization time, with new cells (within the limits of the area of interest) read as required at run time. These additional reads will not require explicit application calls. ICTDB will maintain a list of cells within the area of interest which have been read, and a read of a new cell will be implicitly triggered by database references outside the extents.

ICTDB supports a rich set of feature types and attributes. In order to make the implementation flexible enough to support adding feature types and attributes at run time without a dramatic impact on performance, feature types and attributes have been divided into two categories. First, there are the *core* feature types and *core* attributes for an ICTDB database. Core feature types and attributes are defined via a compiler data file when an ICTDB database is built. Each core feature type has an associated set of attributes taken from the global set of defined core attributes. The core attributes of a core feature type can be inferred from the feature type name. Similarly, the data type of a core attribute can be inferred from the attribute name. In addition to core feature types and attributes, ICTDB supports *extended* feature types and attributes. The attributes of an extended feature type and the data types of extended attributes are explicitly tagged. Extended feature types and attributes may be defined at run time, but access is slower and storage is less efficient.

ICTDB stores topological information about the terrain in several ways. 2-1/2D information about network features such as roads and rivers is topological. The internal representation is an abstract graph to allow non-planar networks of roads. ICTDB stores a *level three*, or full planar topology for TIN data. This means that complete edge adjacency information is stored, although each edge is shared by only two triangles. This mean of course, that three dimensional features which are not *simply-connected*, such as a tunnel, are not supported in the local representation. (In topology, a surface is said to be simply-connected if closed curves drawn on a surface can be shrunk to a point on the surface. A sphere is simply-connected; a torus is not.) In order to support a full 3D topology, features can be placed in ICTDB which have arbitrarily complex geometry, fully supporting multiple elevations with route planning implications. Such features are expected to be rare.

Because full topology is only supported in feature data, all changes to the topology of an ICTDB database must be made by adding or deleting features,

or by modifying existing features. Feature addition requires update to all three database components: local, global and feature data. When a feature is added, all terrain elements in the TIN data which intersect the footprint of the feature are marked. This is necessary since a feature's geometry overrides the geometry of the terrain element(s) on which it is planted. This is required since changes to the underlying TIN are not supported at run time, and features can be added to the database that negate the validity of terrain elements (holes, trenches, etc.) Feature modification means changing the attributes of an existing feature. One frequent case of this will be modifications to the geometry of a feature. If model references or feature references are in use, then a new model or attribute set will need to be generated in the model/attribute library. Feature deletion must be supported to implement retraction in a view (see below). However, deletion from the default ground truth view is not supported.

In ICTDB, no history of updates in maintained. The database represents ground truth as a simulation unfolds. We have not yet addressed the issue of coherence in the presence of multiple assertions and retractions. Since ICTDB does not support retractions in the ground truth view, as explained below, this is not expected to be a problem.

ICTDB does not provide *derived* data, or features, such as mobility corridors. The issue of assertions with side effects on derived features is being addressed in the CFOR Environmental Utilities (MITRE, 1995). Our eventual goal is to fully integrate ICTDB with the CFOR infrastructure and reuse the software components which support assertions (Layer 1) and mobility corridor analysis (Layer 2B).

One of the more challenging desiderata emerging from the ICTDB requirements phase was the realization that a CGF terrain subsystem ought to support multiple database *views*. Views are a standard software layer in commercial database management products. Furthermore, in a CGF application, it is natural to want planning or situational awareness code to be able to make *assertions* about the terrain environment which are notional or temporary, or which have been derived from intelligence information. As an example, it should be possible to reason about "What would be the tactical significance of a bridge placed at location P?" or "How would route planning change if a road segment was added between points A and B?" Since ICTDB supports dynamic terrain, assertions can be made via the same interface function which adds dynamic features to the database. A *retraction*, that is undoing an assertion, can than be implemented as feature deletion. This is the only case we could see in fact, where actual feature deletion would occur.

The default view in ICTDB is ground truth. No attempt is made to save the static view of the database at exercise initialization. A small number of additional views can be created by user request. Each feature has a bitmask specifying view membership on a per-feature basis. In addition, a table mapping feature types to view masks will support the membership of feature types in a view. Thus, all features of a given type can be added to, or deleted from, a user-defined view of the database.

4. Conclusion

We have completed the requirements analysis and design phases of this project, and are in the process of implementing key components and integrating them into ModSAF. We have started with the local representation and global coordinate system. We plan to demonstrate these enhancements to ModSAF in the summer of 1995. The global component and dynamic aspects of the ICTDB representation will be added later in 1995.

5. Acknowledgment

This work is being done as part of contract DACA76-94-C-002 from the Advanced Research Projects Agency (ARPA) and the US Army Topographic Engineering Center (TEC). The authors wish to thank George Lukes of ARPA and Kevin Mullane of TEC for their interest, encouragement, and guidance. We would also like to thank all of the organizations and individuals that have helped with the requirements surveys and analyses.

6. References

Brockway, D., Weiblen, M. (1994), "World Reference Model", Working Document prepared for ARPA War Breaker Program, Release 2 Draft 12.

Buettner, C., Chamberlain, F., Drutman, C., Evans, A., Stanzione, T. (1995), "Integrated CGF Terrain Database Data Source Definition Document", TASC.

Burchfiel, J., Smyth, S., (1990), "Use of Global Coordinates in the SIMNET Protocol", White Paper ASD-90-10, BBN Systems and

Technologies Corporation, Advanced Simulation Division.
Evans, A., Stanzione, T., Chamberlain, F. (1994), "Integrated CGF Terrain Database Application Programmer's Interface Specification, First Draft", TASC.
Evans, A., Stanzione, T., (1995), "Coordinate Representations for CGF Systems", White Paper submitted to ARPA.
Gadeken, L., (1976), "Cartesian Coordinate Transformations in the Elliptical and Spherical Approximations to the Shape of the Earth", MRS3 Working Paper No. 17, PAR Corporation.
MITRE, (1995), CFOR Environmental Utilities API.
Scarlatos, L. (1993), "Spatial Data Representations for Rapid Visualization and Analysis", Doctoral Dissertation, State University of New York at Stony Brook.
Smith, J. (1994), "Compact Terrain Data Base Library User Manual and Report", LORAL ADS.
Stanzione, T., Evans, A. (1994), "Integrated CGF Terrain Database Requirements Analysis, Second Draft", TASC.

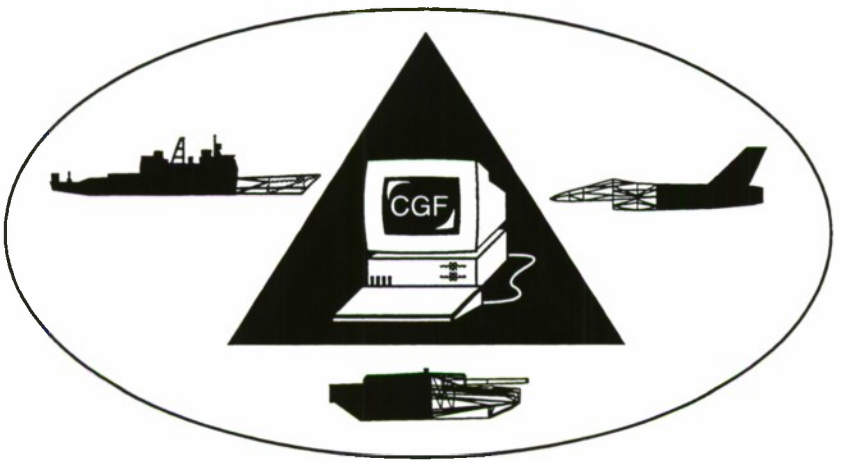
7. Authors' Biographies

Thomas Stanzione is the manager of the Synthetic Environment Section at TASC. He is the Program Manager for the ICTDB project and a key contributor to TASC's other Synthetic Environment programs, including Weather in DIS (WINDS) and Multi-Echelon CFOR with ForeSight (MECFS). Prior to joining TASC, Mr. Stanzione served as the deputy director of the Semi-Automated Forces group at Loral's Advanced Simulation Division (LADS). Mr. Stanzione has a Masters of Science degree in Photographic Science from the Rochester Institute of Technology.

Alan B. Evans is a Senior Engineer in SAIC's Technology Research Group (TRG). Dr. Evans is the Principle Investigator for the SimTool IR&D effort and the technical lead on the ICTDB Terrain Database effort sponsored under the ARPA Synthetic Environments Program. His areas of interest include simulation architecture and performance analysis, terrain representation and reasoning, and tactics and behavioral representation. Before joining SAIC, Dr. Evans served with Bolt, Beranek and Newman's Advanced Simulation Division (ASD), and subsequently, Loral's Advanced Simulation Division (LADS). Dr. Evans holds an M.S. and a Ph.D. in Mathematics from Michigan State University as well as an M.S. in Computer Science from New York University.

Cedric Buettner is a Software Engineer at SAIC. Prior to joining SAIC, he worked on Raytheon's Patriot Fire Unit software developing prototype tactical system enhancements. He is a graduate of Gordon College in Wenham, MA with a BS in Physics and Mathematics.

Forrest Chamberlain is a Member of the Technical Staff in the Signal and Image Technology Division at TASC. Forrest has been involved in Computer Generated Forces work since joining TASC in 1994. Prior to that, he was a critical contributor to the hardware and software design of a "wearable" computer system at Carnegie Mellon University, where he earned his Masters Degree in Electrical and Computer Engineering.


CGF

Terrain Capabilities in CCTT

Jon Watkins
Science Applications International Corporation
3045 Technology Parkway
Orlando, FL 32826–3299
watkins@greatwall.cctt.com

1. Abstract

The Close Combat Tactical Trainer is a complex training system composed of manned modules, user workstations, CGF simulators, and a number of support stations for system initialization and after action review. Each of these components has different requirements for terrain representation and use, including visualization of the simulated environment for trainee immersion, viewing of the terrain on a two dimensional display, basic terrain query routines, and terrain reasoning. In CCTT, these terrain functions are supported by three representations referred to as the visual, plan view, and terrain reasoning databases.

This paper focuses on the Environment CSC, which provides terrain query and terrain reasoning functionality by operating on the terrain reasoning database. Areas of interest include design issues, use by different CCTT components, database representation, and database correlation.

2. Overview

2.1 CCTT Project Summary

The Close Combat Tactical Trainer (CCTT) is the first system in the Combined Arms Tactical Trainer (CATT) family of training systems. CCTT will utilize the Distributed Interactive Simulation (DIS) network protocol to provide a virtual environment for training of armor and mechanized infantry personnel. CCTT is composed of a variety of manned modules, an Operations Center (OC), Semi–Automated Forces (SAF), and several support workstations. The manned modules are cabin simulations with virtual out–the–window views for training on vehicles such as the M1A2, M2A2, and M113. SAF and OC provide emulated vehicles to populate the battlefield; they share a common architecture referred to as Computer Generated Forces (CGF). SAF provides a wide range of both BLUFOR (friendly) and OPFOR (enemy) entities. OC provides BLUFOR entities to support battalion staff training and to add depth to the battlefield with entities which provide resupply, maintenance, combat engineering, and fire support capabilities. Both SAF and OC are controlled via user interfaces provided on the SAF Workstations and OC Workstations, respectively. The actual simulation of the SAF and OC entities is provided by separate CGF processors dedicated to entity simulation.

There are three correlated databases used throughout the CCTT system. The visual database is used for all out–the–window visual displays. The plan view display (PVD) database is heavily optimized to meet response time and display requirements for a two dimensional display on user interfaces in a format similar to standard maps. The "Model Reference" terrain database (or MRTDB) is used for all other terrain operations. The Environment CSC operates on MRTDB (among other objects) to provide terrain query and terrain reasoning capabilities. While the Environment CSC was originally designed to support terrain reasoning operations on CGF systems, other CCTT components use it as well.

2.2 Scope of Paper

CCTT is utilizing spiral development to mitigate risk and provide incremental drops before contract completion. Simple terrain operations were provided in two of the early system spirals in which SAF participated (Builds 2 and 4). As this paper is written, Build 5 integration efforts are underway. Because there are two more builds following Build 5, and because the requirements analysis efforts for these last two builds have yet to begin, the issues and resolutions described in this paper may be subject to change. The intent of this paper is merely to inform the community of current work, thereby providing insight into one aspect of a large and potentially influential system in the simulation and training community. There will continue to be changes and improvements to the terrain capabilities, terrain representation, and requirements as system level issues are resolved and spiral development moves forward. Thus, while the final CCTT terrain implementation may or may not include all ideas presented herein, the discussions still serve to provide practical examples of how CCTT's ambitious requirements are encouraging new approaches to terrain functionality and representations.

2.3 Paper Topics

Section 3 will provide a brief description of the Environment CSC from a functional standpoint. The varying needs of the Environment CSC's disparate users are discussed in Section 4. The following sections then focus on MRTDB, including how the database is generated, representation issues, and a brief overview of the database design.

3. Environment CSC Capabilities

The Environment CSC is the software component responsible for providing terrain querying (e.g. height of terrain) and terrain reasoning (e.g. obstacle avoidance) functionality. One design objective for the Environment CSC is to provide both "sight" and basic "interpretation" of what is seen. This is analogous to the man–in–the–loop looking out the manned modules' vision blocks and observing an undulation in the terrain skin which he determines is a good cover location (without determining, for example, when to go to the cover position). The Environment CSC, then, encapsulates basic environment data (such as the terrain database representation) by providing "value added" routines to callers.

Functionality developed through Build 5 includes height of/above terrain, collision detection, munition impact detection, line of sight, road routing, static obstacle avoidance, area intervisibility, and covered positions based upon terrain skin. Additional functionality to be provided in Builds 6 & 7 includes cross country routing, refinements to obstacle avoidance, cover and concealment based upon features, and weather. These and other future capabilities are discussed from the perspective of a "snapshot" of the latest available requirements and preliminary design.

While many of these capabilities will sound familiar to those who have examined other CGF systems, a number of new twists have been provided by CCTT's databases and requirements. As a result, significant new functionality is provided as discussed throughout this paper. A few sample cases are briefly discussed here.

3.1 Sample Issue: Dynamic Terrain

Some terrain features (referred to as prepositioned objects), will change their geometry at run time. This requires a mechanism for uniquely identifying features and a means to efficiently alter their geometries. In addition, Combat Engineering emplacements (referred to as relocatable objects) may be created before and during the exercise for survivability, counter–mobility, and mobility operations. Relocatable objects may supplement the geometry of terrain skin, as well as create new obstacles where none existed before. Dynamic terrain issues for CGF and DIS representation are discussed in (Campbell 1994) and (Crowley 1994).

One interesting issue presently being addressed in CCTT is how to handle "omniscience" relative to terrain state. If a bridge is destroyed by an OPFOR entity, BLUFOR entities should not automatically alter their long distance road routes if there was no mechanism by which they could have known that the bridge was destroyed. However, other OPFOR entities that could have communicated with the entities who destroyed the bridge should respond appropriately. Finally, when BLUFOR entities first "sight" the destroyed bridge, they should be able to inform other friendly entities of the bridge's destruction.

Design discussions on this issue are underway, but the Environment CSC presently will provide routines which will allow a caller to create and modify a private data type representing an "awareness state". Because it is up to the Environment CSC's consumers to create and retain these structures, support is provided for variable levels of "awareness" resolution. CCTT will probably use a simple "force" case wherein each side is a collective conscience such that what any OPFOR entity knows about the terrain state is known by all other OPFOR entities. If future needs require a unique awareness for each company, platoon, or even entity, this is already supported by the Environment CSC, since any number of awareness state data structures may be created and operated on.

Observe that only a subset of Environment CSC capabilities are impacted by the awareness issue. Query operations are inherently based on reality rather than perception (e.g. line of sight is blocked by a log crib regardless of who is aware of the log crib's presence). Long distance routing is the main piece of functionality which must accept a terrain "state" from the caller in order to determine what is known. Obstacle avoidance is impacted, but only for minefields because other obstacles such as log cribs and destroyed bridges can be seen when an entity gets close enough to plan a local path. Thus, obstacle avoidance can accept the caller's perceived state and avoid only those minefields which the CGF entity is aware of, while correctly blundering into the others.

3.2 Sample Issue: Overpasses and Bridges

Bi–level terrain will be supported in CCTT for overpasses and bridges. Support will extend beyond recognition of multiple valid Z values at a given x,y loca-

tion. Bridge supports running vertically from the ground to the suspended span are recognized by collision detection, munition impact detection, line of sight, obstacle avoidance, etc. Suspended spans will be recognized by these same algorithms as well as by cover and concealment, providing overhead cover for entities. For destructible bridges, munition impact detection will uniquely identify the bridge so the detonation PDU can name the bridge explicitly. Destroyed bridges will result in the entire suspended span being removed. System design decisions require us to consider the suspended roadway "sides" to be a collision volume so that entities cannot maneuver off the suspended parts of a bridge or overpass.

3.3 Sample Issue: Penetrable Forests

Original system specifications required tree densities of approximately 150 trees per square kilometer as a representation of forested areas. Customer desires and innovative applications of image generator (IG) resources permitted much higher fidelity representations of forests than the aforementioned density of trees or SIMNET's familiar "canopies". Evans & Sutherland was able to provide incredible tree densities by using "fading" boundaries (on forest sides and tops) that provided the illusion of many trees while actually only displaying relatively few at a time. Unfortunately, the tree density was so high as to overwhelm any practical CGF representation. In addition, representation of the fading boundaries would have tied the Environment CSC's capabilities to a specific IG technique for load management, something we have sought to avoid.

A number of system tradeoffs were discussed, including use of an abstract "fog" to represent the interior sections of forests in CGF, serious reductions in tree densities, and "bands" of variable density. More background on options pursued is provided by (Braudaway 1995). The final solution agreed upon provides a relatively high tree density (peaking at a density of about 2200 trees per square kilometer), that will nonetheless be fully correlated in MRTDB. The IG will dynamically introduce forest boundaries at 1 km in support of load management, but demonstrations indicated that there is no need for CGF to represent these artificial boundaries because the tree densities at forest edges are so high that the transition from trees to boundary is difficult to perceive at a range of 1 km. Thus CGF represents forests "correctly" (i.e. as many individual trees), without dealing with IG techniques such as boundaries or canopies. This provides extensibility because as IG capabilities improve over time, the boundary range or tree densities may increase, but this will not require changes to CGF's representation.

As part of the give and take of system design, the penetrable forest does include some "stretch" for MRTDB and Environment CSC algorithms. The tree densities are sufficiently high so as to threaten available caching space due to potential spikes in storage requirements in certain regions, and would also sharply increase the total size of the database. Even with a tightly space–optimized representation, storage of the 10 million independent tree instances that are expected to appear in CCTT's forested Primary #1 database could consume 160 megabytes.

One possible approach to resolving this issue is the use of "aggregate models", wherein groups of trees whose 2D configuration is used repeatedly throughout the visual database are stored once and referenced many times with different offsets. We have investigated ideas that will maintain the low cost 2D filtering that is at the heart of efficient feature accesses for terrain operations. However, 2D placement of each tree surviving these initial filters will require 2 integer additions, and Z placement will require several floating point multiplications and additions. This cost may be acceptable for the forest interiors where expensive line of sight operations (which will require full 3D placement of trees) will typically be truncated at very short ranges due to the high tree densities Other algorithms either don't need a full 3D representation (e.g. obstacle avoidance) or can use 2D filtering to limit full tree placement (e.g. collision detection). We plan to experiment with aggregate models in Build 6.

4. Non–CGF Requirements

Originally, MRTDB and the Environment CSC were designed for use on CGF simulators (which use most CSC capabilities) and SAF workstations (area intervisibility, routing support, etc.). Later, though, it became apparent that other CCTT components could make use of these CGF–oriented capabilities. For example, the after action review stations use the Environment CSC to keep the "stealth" eyepoint within the database dimensions and above the terrain skin. Also, scenario analysis and reporting information can be extended beyond what is available directly from DIS traffic, such as determining line of sight between various entities. Also, the OC workstation (run by a trainee) has slightly different needs than the SAF workstation (run by a dedicated operator) which can also be met by the Environment CSC (e.g. different levels of support for automated routing can be achieved with the same software).

The aforementioned CCTT components have been able to reduce development efforts by making use of software originally developed for CGF; however, the greatest benefit has been derived from manned modules' (MM) reuse of the Environment CSC.

4.1 MM vs CGF: Correlation Issues

While the DIS standard has gone a long way toward providing interoperability between fundamentally different simulation components in a networked environment, there are a number of other interoperability issues that must be dealt with. Database correlation is an oft–cited example of such an issue. Another layer on top of this problem is the possibility of different applications having a different perception of the world not because of correlation errors in the databases themselves, but rather in the algorithms which are accessing and operating on the databases.

The issue of correlation between the visual and CGF databases has been addressed aggressively in CCTT, and appears well in hand despite a number of pitfalls encountered along the way. Significant work remains before us in this area. However, CCTT has managed to shift the related problem of algorithms operating on the different databases up a layer in the conceptual "protocol stack" of terrain correlation, by having CCTT manned modules use a subset of the same Environment CSC used by CGF. The subset of Environment CSC capabilities needed by manned modules are height of terrain at a location, mobility indication (i.e. surface characterization) at a location, collision detection, munition impact detection, and a number of simple query routines (e.g. indication of whether an area is "forested", "urban", or "open" in support of damage assessment from proximate impacts). This idea represents a departure from the original CCTT design which called for use of the IGs for basic terrain operations needed by manned modules. The correlation problem is thus reduced to just data correlation without the additional concern of access algorithm correlation. If a manned module queries the height of terrain at a given x,y location it will get the same z result as a CGF entity at the same location.

While the Environment CSC provides a self–consistent baseline for CGF and manned modules, these two systems have somewhat different requirements and needs which drive them to use the supplied information in different ways. At times, we have been able to meet all needs with a single (more generic) interface, but sometimes specialized access routines were required at the external interface level.

4.2 MM vs CGF: Vehicle Placement

Vehicle placement requirements represent an example where our interface was impacted by the different needs of our consumers. Based upon legacy systems, an interface for vehicle placement (returning, for example, a rotation matrix) may have been provided. Our object oriented design pushed us away from this concept (i.e. a rotation matrix is an attribute of an entity, which has a heading, not of the terrain), and this notion was reinforced when it was discovered that manned modules needs up to 14 contact points, each with a unit normal to the terrain skin and a mobility indicator. In contrast, CGF's simpler entity placement requires only four contact points. While the Environment CSC uses the same primitive routines for determining elevation and mobility indicators, a specialized external interface was added for manned modules so the unit normal could be returned to them without complicating CGF's interfaces.

4.3 MM vs CGF: Collision Detection

For collision detection, CGF uses a single, fully oriented bounding volume to describe the location of the querying entity. However, manned modules requires separate bounding volumes for the hull, turret, and gun. In this case, the Environment CSC interface simply became more generic by providing the caller the opportunity to specify the bounding volume to be operated against. In this manner, the caller can select the fidelity of representation to be used without changes to the collision detection code itself.

4.4 MM vs CGF: Caching Approaches

Manned modules' caching needs have a clear bound since a region need only be maintained around the own–vehicle in order to prevent cache misses. A simple look–ahead caching scheme can maintain the cached area around the own–vehicle at sufficient distance to handle worst–case munition flyout ranges. However, manned modules' extremely stringent timing requirements (based upon a required 15 Hz update of the IG) leave no leeway for cache misses or even disk I/O needed for look–ahead cache priming.

At the opposite end of the spectrum, CGF must deal with many vehicles, potentially spread out over large areas. While this complicates efforts to predictively read in sufficient terrain data to avoid a cache miss during a terrain query, CGF is nonetheless more tolerant of a missed update for a given entity (or component of an entity) as long as other entities can still be updated.

Use of a separate AIX process for disk I/O when accessing MRTDB goes a long way toward meeting these

different needs. While disk I/O is blocking the caching process, the main process can continue on. This may resolve the problem of manned module's strict update times since the 15 Hz appointment can be met even while disk I/O for predictive caching is underway. The separate process simultaneously meets CGF's needs because a cache miss for critical data can cause an entity component to "yield" its tick. This allows other entities to continue on unaffected, while the terrain data should be available for the next tick of the skipped component. This works well for some terrain requests (line of sight, path planning), but may cause anomalies for other capabilities (vehicle dynamics, collision detection). These are the same anomalies that would be encountered on a global scale as any CGF system's performance degrades under peak load. The alternative would be to hold the entire CGF process during disk I/O, thus imposing the worst–case performance on all entities on the affected CGF processor. These issues will be explored in more detail during Builds 6 & 7, including decisions as to whether it is practical to implement the concept of "yielding" a tick.

4.5 MM vs CGF: SIMNET Interoperability

The final requirement levied on the Environment CSC in support of manned modules was correlation of and operation on the Grafenfels database, which is needed for SIMNET interoperability. While CCTT manned modules (and other components) are required to interoperate with SIMNET (via a protocol translator), the CCTT CGF system is not required to do so. Thus, under the original system design of using the IG for manned module terrain needs, there would have been no need to generate a terrain reasoning version of the SIMNET interop database. CCTT databases represent a superset of SIMNET databases with the exception of canopies and treelines (as discussed elsewhere in this paper), so these were the main problem areas. Fortunately, SIMNET treelines and canopies do not effect any of the functional areas needed by manned modules (e.g. these features don't cause collisions in SIMNET). As a result, only minimal effort was required to support SIMNET databases for manned modules. One simple example is the fact that tree trunks also do not cause collisions in SIMNET, so we must provide a mechanism for ignoring tree trunks while determining collisions in SIMNET interoperability mode.

5. Terrain Database Generation

The process for deriving the terrain reasoning and PVD databases from the visual database source data is illustrated in Figure 1. The visual database is built from a number of sources to meet the needs of the customer while simultaneously conforming to polygon budgets for the target IG. The resultant database is then exported to a data file (which is in a modified SIF format, aka "SIF++"). The details of the data source format are hidden by an API, which is linked in by any application which needs to build a correlated database. This API provides data in a consumer–oriented format (in contrast to a format reminiscent of the data source), with some value added to simplify the consumer application and reduce duplication (e.g. regional access and clipping).

This approach can be extended for support of other data sources by simply replacing the CCTT API with a version that accesses the new data source. The new data source could be the S1000 API, CTDB, or any other terrain representation. By the same token, other target database formats could be generated by creating a new terrain compiler that links in the CCTT API.

6. Terrain Representation Issues

In addition to the functional issues described in Section 3, a number of design concerns arise directly from the nature of the terrain database being generated in support of visual requirements.

6.1 Terrain database density and size

The density of terrain features and resolution of terrain skin are being driven by the substantial capabilities of the ESIG HD, which is the CCTT visual system (Evans & Sutherland 1994). The terrain skin will be represented with minimum base ground polygon sizes of 30m for CCTT's desert database ("Primary #2"), and of 60m for the forested database ("Primary #1"). Microterrain (also known as "cut and fill") will be used extensively to fracture or build up the base terrain polygons to ensure that all roads are trafficable and that rivers don't appear to flow uphill.

Visual database development is still underway and a number of fundamental design issues are still outstanding; however, initial estimates indicate that over 30,000 man–made structures (10,000 of which will be destructible) and over *10 million* individual trees will be present in the first full–size CCTT database. These overwhelming feature densities in databases of 100 km x 150 km have forced us to design around the idea that only small subsets of the database will be accessible in memory at one time and that it will not be practical to read in all of the terrain needed for long range operations such as radio degradation and fixed–wing aircraft terrain following. Some examples of our approach to these issues are provided in 3.3 and 7.4.

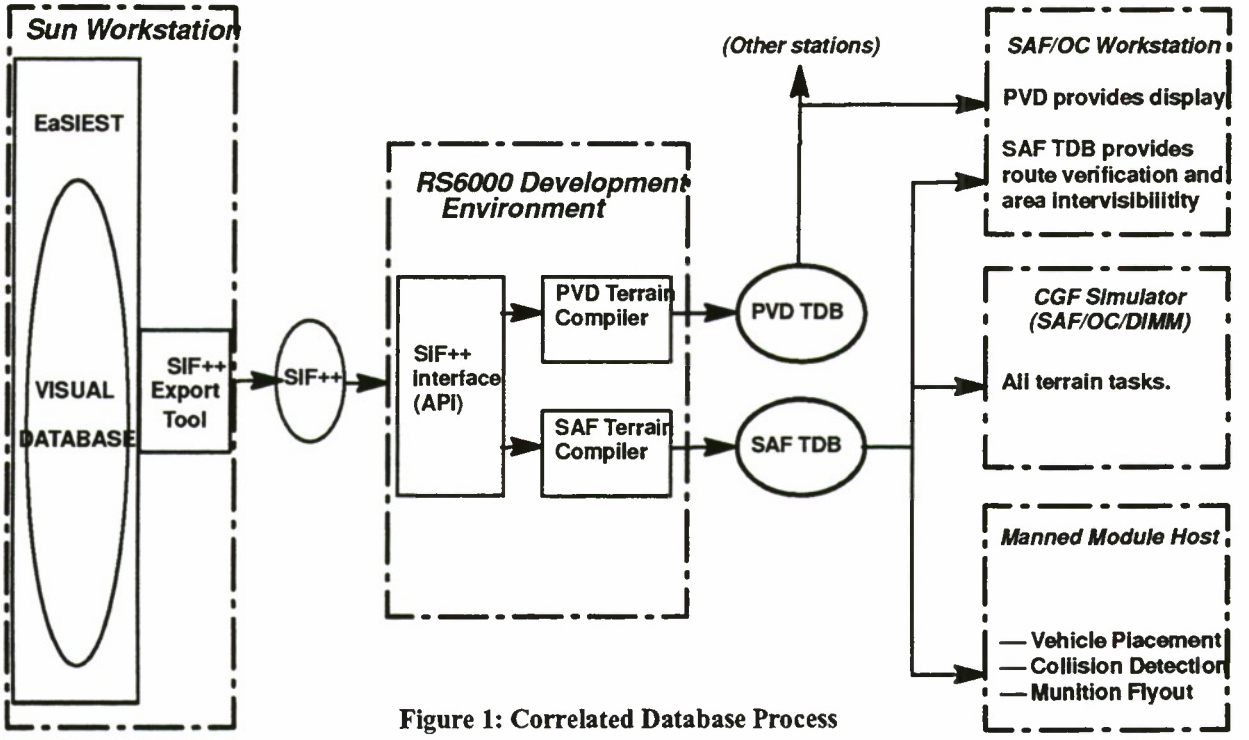


Figure 1: Correlated Database Process

Point features (trees, buildings, bridges) will be much more dense than in SIMNET databases. Indeed, feature densities will be sufficient to provide detailed representations of urban areas and forests. Since tree densities can be greater than 2200 trees per square kilometer, two familiar SIMNET features types, canopies (hollow, tent–like structures representing forests) and treelines (paper thin green "walls" representing stands of trees), are no longer needed.

6.2 Detailed Representations

A number of CCTT requirements are driving us to consider more complex or detailed feature representations than would otherwise be used. For example, CCTT entities are required to detect collisions with tree trunks as opposed to foliage, while foliage affects line of sight, thus requiring us to store a radius for the trunk and a radius for the foliage.

There will be approximately 16 to 30 unique mobility indicators in the terrain reasoning database. These mobility indicators will be determined via a mapping from those visual database "material codes" which may be driven on. The "drive on" material codes are essentially permutations of ITD thematic layer values. We must also support a mapping from a "dry state" material indicator and its corresponding "wet state" indicator in order to provide different mobility effects in rainy conditions. The material code boundaries will be conveyed as complex, dense areal features which need not conform to post boundaries. The number of points expected for these feature types will defy full representation given the timing and storage requirements for manned modules and CGF, so we are investigating approximations that provide a balance between these performance concerns, the customer's desire to have a reasonable range of unique mobility effects, and our ability to provide sufficient correlation with the visual database.

As alluded to earlier, accuracy requirements for collision detection and munition flyout capabilities have driven us to consider use of bounding volumes for entities instead of potentially faster representations such as "leading edge" or "sample point" calculations for collisions. Use of more expensive routines for high fidelity calculations encourage the introduction of additional layers providing filtering in order to reduce the number of high fidelity calculations required, especially in light of the unusually high feature densities in CCTT databases. This filtering is beyond the fast spatial filtering provided by the primitive terrain database access routines and supported by the database representation. For example, collision detection can use a rectangle orthogonal to the coordinate system axes defined by the two–dimensional minimum and maximum points of the fully oriented bounding volume as a very fast filter mechanism, which should reduce or

eliminate complex checks against excessive numbers of nearby features. These types of algorithmic filters are made more important by the sharp peaks in feature densities exemplified by penetrable forests.

6.3 New feature types

Hedgerows, walls, dams, suspended bridges, and overpasses are all presently planned for the visual database. Representation of tunnels is still under discussion. The visual database's representation of forests has allowed incredible tree densities to be represented (design issues with penetrable forests are expanded on in 3.3). Urban areas are represented with high densities of buildings complete with urban clutter, driveways, and residential streets.

6.4 Storage and Correlation

The Environment CSC uses a number of approximations and simplifications in both algorithms and terrain representation in an attempt to balance fidelity, performance, and correlation concerns. We are investigating low–cost solutions to the correlation errors derived from some of the simplified feature storage mechanisms we have reused from other database formats.

One example is the truncation effect caused by storage of features in square patches with local coordinate systems bounded to the patch's dimensions. Trees sitting at the boundary of a patch would appear to accessing routines to be truncated at the patch edge. Other feature types, including roads and rivers would suffer the same problems. In some cases, one can store pieces of the feature in each patch. This would complicate abstract recognition of features. For example, a building sitting at the intersection of four patches would appear as 4 buildings, thus confusing reasoning on the feature while also complicating operations on uniquely identified destructible buildings. In other cases, storage of multiple clipped features would require special case code, such as storing most of a river in one patch, and the remainder in another. This case would confuse code which assesses a river feature to see if an AVLB can span it; perhaps each of the truncated features can be spanned, while the actual full–size river cannot.

If linear features are stored as a series of line segments and widths, then correlation is sacrificed at each "bend" on the linear. The error becomes more significant for wider features and sharper turns. The gaps which appear in linears may be marginally acceptable for roads, but significant errors in river linears may result in unacceptably anomalous behavior.

Figure 2 illustrates these concerns. The four boxes represent patches. Feature 1 is a linear (i.e. road or river) which has small "truncation" errors at the top and bottom. Feature 2 is a building which crosses 4 patches. Feature 3 is a tree which cannot be accurately represented by duplicate feature types (i.e. a point and radius in the upper patch would not match the needed geometry). Feature 4 is a linear which would suffer truncation on the vertical segment and also demonstrates some representation errors on extreme turns.

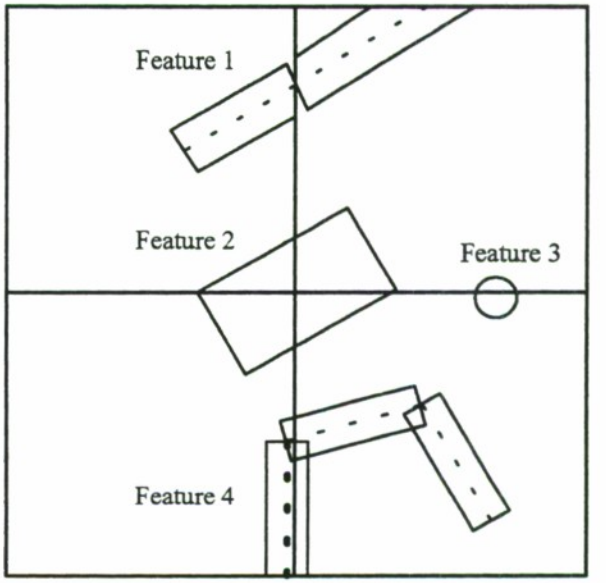


Figure 2: Feature Truncation & Linear Storage

7. MRTDB Design

This section provides a brief overview of some of the design principles being used or experimented with in MRTDB's design. These ideas are a snapshot of the Build 5 representation, along with some speculative information on future capabilities such as dynamic terrain. A much more extensive treatment of these concepts as applied to the Build 4 MRTDB version is provided by (Watkins 1994).

7.1 Object Oriented Design

We used an object–oriented approach in designing and implementing the Environment CSC and MRTDB; see (Rumbaugh 1991) for the OOD methodology used within CCTT. We designed our database using Rumbaugh's object–oriented design methods and implemented it in the Ada language. Figure 2 is the terrain database Object Model using Rumbaugh's Object Model notation.

7.2 Separate Files

As discussed in Section 4, different consumers with different needs use the Environment CSC. One early concern with this reuse was the fact that all CCTT

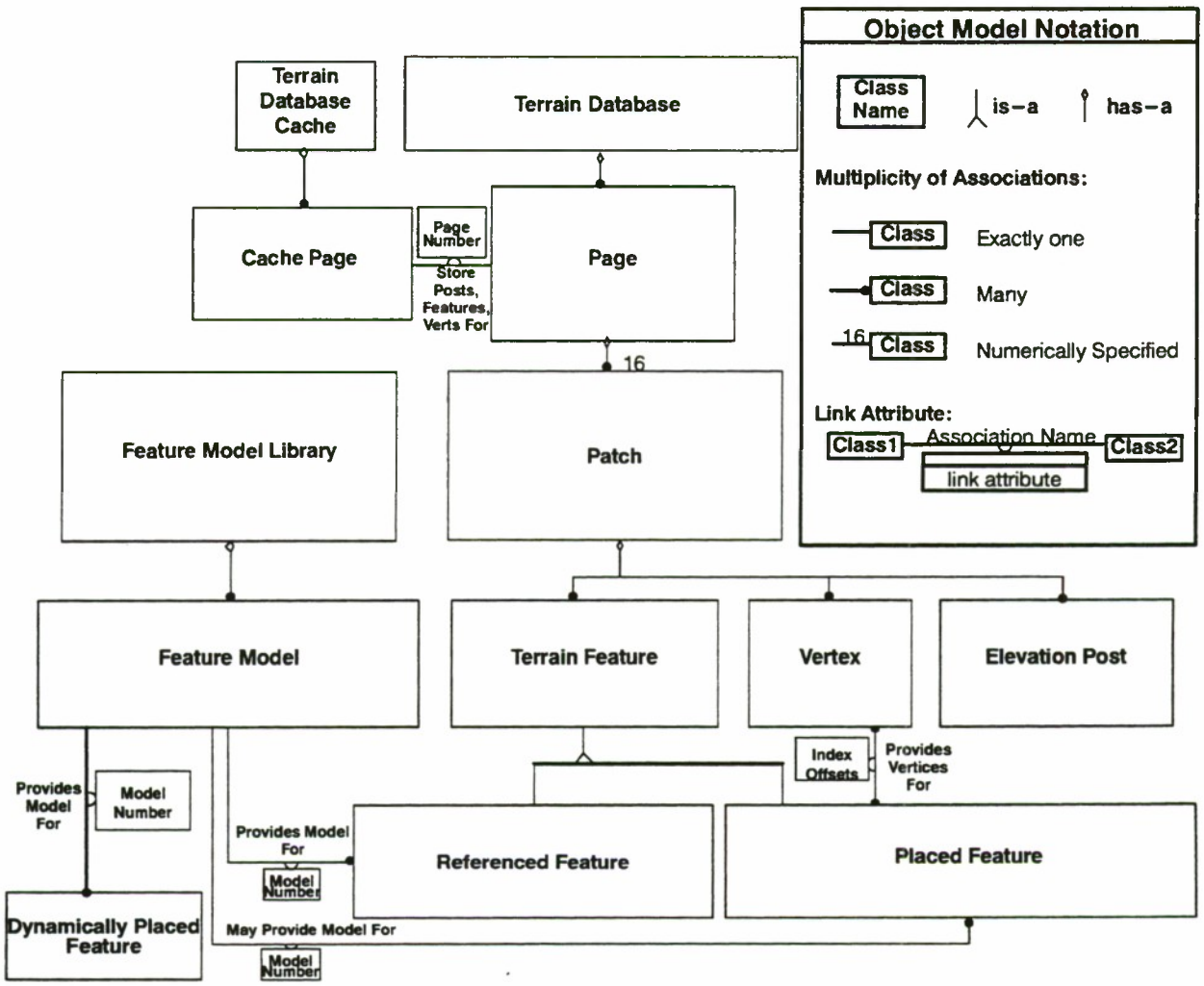


Figure 3: Terrain database object model.

components using the Environment CSC might need to store the massive database file used for CGF terrain reasoning. While estimates of final database size are difficult at this point, the larger of CCTT's two database (Primary #1) may be between 150M and 350M. To minimize the storage impact on the Environment CSC's users, the terrain database is composed of a number of files. Thus, OC workstation, which only needs routing support, does not need to read in (or even store on its hard drive) the 100's of megabytes of high fidelity terrain data. The space savings for applications which need the high fidelity representation but not routing (e.g. manned modules) is less, but it is still useful considering how important sufficient cache space is.

A potential benefit of storing a database as multiple files is incremental database development and/or selective replacement of portions of the database. For example, if something changes in the header file, the much larger feature file need not change (and thus feature offset values need not change, etc.). Replacement of the model file is discussed in the following subsection.

7.3 Feature Model Library

The Feature Model Library is a set of feature models, where each model has a unique model ID. Each model maintains information about a feature that is common across many features. For instance, a tree model maintains the opacity, height, and radius of the foliage, as well as the trunk radius for a particular kind of tree. Each tree in the database that shares the same attributes can then reference the same tree model (via the model ID) to complete its definition. Since the model is stored only once, an enormous space savings can be

realized. In the case of trees, there may be 10's of models in the forested database, but there may be as many as 10 million tree instances.

The Feature Model Library facilitates the implementation of prepositioned objects since to "damage" a building we can simply modify the building model ID from the "normal model" ID to the "damaged model" ID. This is also much faster than performing computations to alter the geometry. Also, use of the library allows for a database modeler to supply the exact model of a damaged building, instead of describing the alterations that would need to be performed on a normal building to damage it. In addition, different buildings may be altered in different ways based on their damaged models, with no additional special case implementation.

Finally, since the Feature Model Library is stored in its own file, it may be swapped out for another library with the same feature models where some number of the feature models contain different information. For example, if we want all the trees to drop their leaves, we can read in a different Feature Model Library where all tree foliage opacities have been diminished. Furthermore, we can increase the fidelity of feature representations with little impact on storage requirements and with disregard for feature densities, since we only need increase the size of the models, not the size of each feature instance.

7.4 Header Data

The Environment CSC relies upon information stored for each patch in the terrain database (where a patch is a square region of terrain which may be scaled depending upon the database). These patch headers are then maintained in memory at all times to provide abstract or low fidelity information for the patch's region of terrain. For example, the off–line database generation tools can apply parametric criteria to these square regions to determine if they are generally urban or forested. A rough characterization of the terrain is provided by storing the maximum elevation in the region. This can then support rough communications degradation or terrain following for fast moving aircraft without requiring reference to the much larger high fidelity representation. We thus avoid the need to cache in huge amounts of data for a long distance communications check.

MRTDB patch headers also contain direct indices into their feature array. In order to determine if a given x,y point is on a road, one need only calculate which patch the point is in, then directly access the road linear features.

8. Conclusions

The Environment CSC, operating on MRTDB, provides a number of key capabilities to CCTT components with dissimilar needs and objectives. Thus, the Environment CSC not only meets CCTT's stringent requirements as implemented to date, but also exemplifies reuse of complex software components in large systems. This reduces development costs and improves correlation. Although many complex issues remain ahead, work to date indicates that each will be resolved in a manner which simultaneously advances the state of the art in distributed simulation and meets the needs of CCTT as a production system. It is hoped that this success will continue not only through the completion of CCTT, but throughout the development and fielding of future CATT systems.

9. References

Braudaway, W. (1995) "CCTT System Architecture Design Note #52: Penetrable Forest".

Campbell, C., McCulley, G. (1994). "Terrain Reasoning Challenges in the CCTT Dynamic Environment", *Proc. of the Fifth Conference on AI, Simulation, and Planning in High–Autonomy Systems*, pp. 55–61.

Crowley, J., Moore, R. (1994). "Synthetic Environment Protocol", *Summary Report of the 11th Workshop on Standards for the Interoperability of Defense Simulations*.

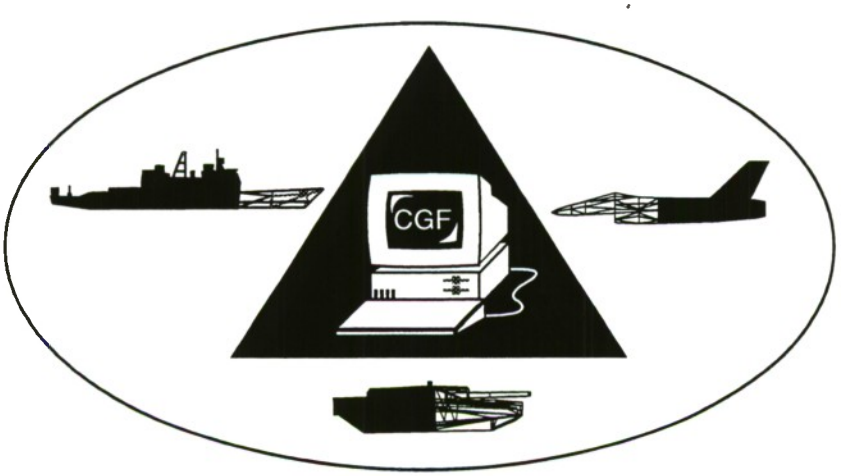
Evans & Sutherland (1994). "Close Combat Tactical Trainer: Central United States", Database Design Document.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenson, W. (1991). *Object–Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey.

Watkins, J., Provost, M. (1994). "Design of Terrain Reasoning Database for CCTT", *Proc. of the Fifth Conference on AI, Simulation, and Planning in High–Autonomy Systems*, pp. 62–68.

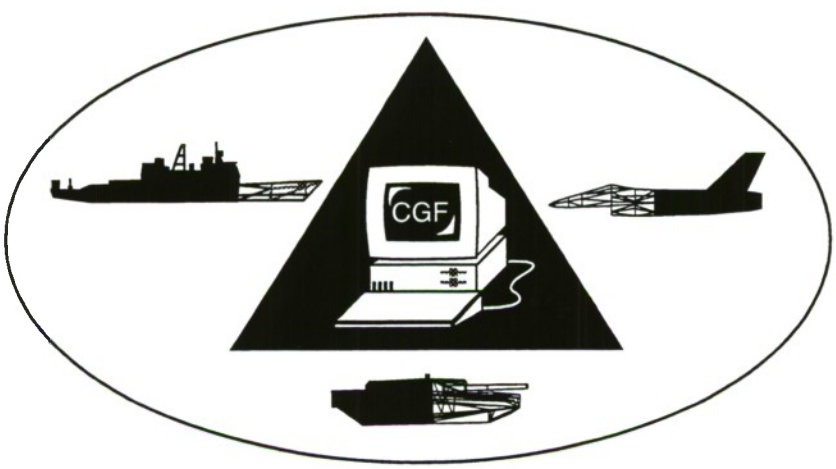
10. Biography

Jon Watkins is team lead for CCTT's "terrain team", focusing on the Environment CSC and several other components. Mr. Watkins received his B.S.E in Computer Engineering from the University of Central Florida in 1990. He has been involved with research and development in the CGF area for nearly five years.


CGF

Evening Plenary Session - ARPA CFOR Briefing

Salisbury - The MITRE Corporation


CGF

Implementation of Command Forces (CFOR) Simulation

Marnie R. Salisbury (marnie@mitre.org)
Lashon B. Booker (booker@mitre.org)
David W. Seidel (dseidel@mitre.org)
Judith S. Dahmann (jdahmann@mitre.org)

The MITRE Corporation
7525 Colshire Drive
McLean VA 22102

1. Abstract

The command forces (CFOR) program will implement a new aspect of warfare simulation: explicit modeling of command and control. The program presents several aspects: (1) a concept of operations where command and control nodes occupy the battlespace in the same manner as weapons systems; (2) an architecture where software simulation of command and control interacts with the battlefield through a set of common services; (3) a software design for the services that forms an infrastructure that integrates with the underlying ModSAF wargame; (4) a mechanism that facilitates automated integration of real world C2 systems with simulations; and (5) an implementation plan that integrates the efforts of multiple developers to produce a functioning multi-service command forces simulation.

The CFOR program has passed through the concept and planning phases and is now beginning implementation. Lessons learned from progress to date are presented along with a plan for multi-vendor implementation.

2. Background

The Command Forces (CFOR) project is a part of the Synthetic Theater of War (STOW) program, an Advanced Concept Technical Demonstration (ACTD) that is jointly sponsored by the United States Atlantic Command (USACOM) and the Advanced Research Projects Agency (ARPA). The STOW program is scheduled to support a USACOM exercise in 1997. In the exercise, objects from each US armed service will interact with each other and with credible opposing force objects in the virtual simulation environment using the Distributed Interactive Simulation (DIS) protocol.

The STOW ACTD requires the ability to represent larger-scale and more diversified military operations in virtual simulation. A key element in achieving this goal is the ability to represent both fighting forces and their commanders in software. Current computer generated forces (CGF) implementations allow the virtual battlefield to be populated with a useful collection of combat entities at the individual platform and small unit levels. CFOR extends the basic DIS architecture to incorporate explicit, virtual representation of command nodes, C^2 information exchange, and command decision making.

3. CFOR Concept

Extension of DIS to incorporate command and control is based on four fundamental tenets.

(1) Command and control can be represented in terms of the interactions and behaviors of command entities.

(2) The C2 process is an information flow process among command entities. As a part of the CFOR concept, the Command and Control Simulation Interface Language (CCSIL) represents the information exchanges between commanders.

(3) C2 information flow must be restricted by a faithful representation of real world communications. Information flow must be routed through command nodes compatible with the real world and subjected to battlefield effects. As with real commanders, virtual command decision makers will have access to information about the world through their sensors, information reported by subordinates through CCSIL messages, and CCSIL intelligence messages from superiors.

(4) The C2 decision process is represented in the individual command entities—the originators and recipients of information exchanges.

Under the CFOR architecture, a command entity may be represented in one of three ways:

- a complex software application (the original goal of the Command Forces program),
- a traditional computer generated forces application (e.g., an abstraction of the platoon leader is embedded in the ModSAF application),
- a human working at his/her real world command

and control workstation.

ARPA's CFOR program is working to build and integrate several examples of all three representations of command entities to create a robust and intelligent synthetic force for the STOW ACTD.

4. CFOR Architecture

The DIS protocols define a common interface for each entity that attempts to ensure interoperability and consistent physical interactions on the virtual battlefield. Analogous requirements exist for the C^2 interactions among entities. Accordingly, the CFOR framework includes an architecture for command entities that extends beyond the DIS network interface to provide a well-defined, common interface for all command decision activities. Command entities are under no implementation constraints beyond those imposed by the interface specified by the architecture. In this way, the CFOR framework extends the basic tenets of the DIS paradigm into a new mode of entity interactions.

The CFOR architecture comprises two primary elements: a technical reference model for command entity development and the Command and Control Simulation Interface Language (CCSIL) for insuring interoperability among

4.1 Command Entity Technical Reference Model

A technical reference model (see Figure 1) was defined that describes the command entity architecture. This architecture promotes interoperability and coherent C^2 activity by providing a shared infrastructure, a common set of information and computing services, accessible through a well-defined applications interface. The architecture is composed of three layers: Application Layer, Information Services and Utilities Layer, and Baseline Infrastructure Layer. A layered approach was selected for three specific benefits: 1) it provides a means of centralizing control over the baseline of doctrinal knowledge needed by the command entity applications; 2) it reduces command entity developers' efforts by providing common reusable software; and 3) it shelters the command entity developers from technology and functional enhancements in the baseline applications (e.g., ModSAF) and allows them to focus on command decision behavior.

- The Command Entity Application layer is where the command decision-making processes reside. Command Entity Applications may be fully automated software or C2 workstations operated by live command entities. All details about the actual implementation of a full automated software command entity are under the purview of the simulation developer organizations; they are free to implement their own approach to making command decisions. Likewise, the adaptation of C2 workstations to the CFOR architecture is dependent only on the interface specification to selected modules with the Information Services layer. Workstation developers have free rein to decide how to display, massage, or augment the simulation data available via the Information Services layer.
- The Information Services layer contains the services and utilities that provide the information needed to support command decisions. These services impose few restrictions on how to model the decision process. They avoid making any inferences or judgments that are the proper purview of command entities.

  Access to the services and utilities is implemented using an object-oriented, implementation-language independent interface between command entity applications and the information services. To accomplish this, the Interface Definition Language (IDL) specification of the Common Object Request Broker Architecture (CORBA) was selected to define the interface and specify all interface parameters.

  Services available include the following:

  - *Platform Behaviors* provide a generic interface to a command entity's physical representation on the battlefield. Services provided mimic the commander's ability to sense from his vehicle,

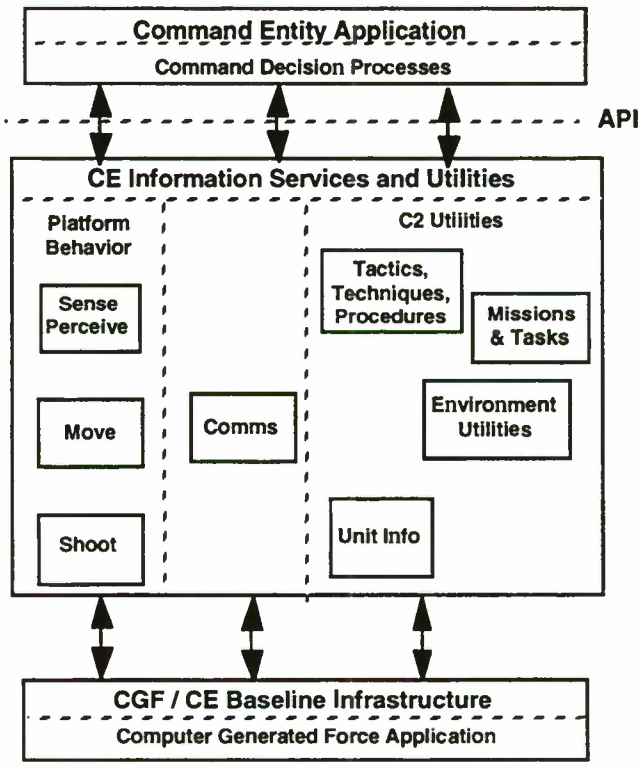


Figure 1. Command Forces Technical Reference Model

move his vehicle around the battlefield, and employ his weapons.

- *Communications* offer an application interface to CCSIL message utilities.
- *Command and Control Utilities* represent the background knowledge and rote reasoning capability of the commander. They include

  Environmental Utilities include the ability to compute mobility corridors, control measures, reverse slopes, routes, travel time and speed. (Environment includes terrain, ocean, and atmosphere.)

  Unit Info provides access to static data about units (own and enemy) and the ability to make basic inferences (e.g., combat power) from the raw data.

  Missions and Tasks which provides doctrinal decision templates to help interpret an ordered mission and to devise a plan.

  Tactics, Techniques, Procedures which provides templates to help fill out orders and implement a plan.

- The Baseline Infrastructure Layer contains the basic platform representation and general DIS interface utilities. These capabilities are accessed by command entity applications indirectly through the Information Services layer.

4.2 CCSIL

The Command and Control Simulation Interface Language (CCSIL) is a special language for communicating between and among command entities and small units of virtual platforms generated by computers for the DIS environment. CCSIL includes a set of messages and a vocabulary of military terms to fill out those messages. CCSIL was developed to facilitate interoperability between various implementations of command entities (i.e., decision makers) and platform entities (e.g., vehicles, weapons, sensors) in the DIS environment.

Figure 2 shows a view of the CFOR architecture from the CCSIL perspective.

A common language designed for interpretation by software (e.g., simulations or cognitive processing systems) is needed to allow all three implementation approaches to work together in one environment. By using the highly structured format of CCSIL messages, humans at real world command and control workstations can send orders and directives to software command entities and expect them to react appropriately. Likewise, software command entities can exchange messages with each other.

Without a common language and communications services, every new element added to a DIS exercise would need to be iteratively retrofitted to interoperate with every other existing element of the virtual simulation environment. CCSIL serves as a unifying thread among diverse implementations of command entities, computer generated forces, and command and control workstations.

5. Integration With Real World Command and Control Systems

"Simulations should be driven by military personnel using their go-to-war C2 systems." Simulation developers, especially in the training simulation world, have heard this requirement expressed routinely by the military user community. Until recently it has been very difficult to meet this requirement. In limited cases, special automated links have been developed to link a particular C2 system with a particular simulation. Unfortunately, these point solutions are not generalizable to other C2 systems or other simulations.

The ability to interface C2 systems with simulations is premised on several characteristics of computer simulations:

1) There needs to be a way for users at real world systems to communicate with simulated counterparts. This means that simulations must represent information exchanges internally in a way functionally compatible with the real world and the simulations must include representation of command functions for the real world users (e.g., commanders and staffs) to communicate with. Most combat simulations have

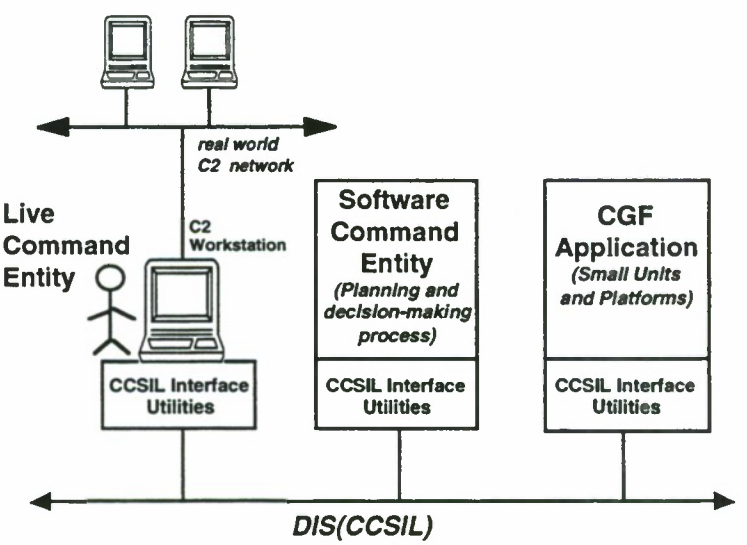


Figure 2. A View of the Command Forces (CFOR) Architecture

not included representations of either C2 information exchange or command entities carrying out the command and control process to produce behavior in the unit. Often combat units are manipulated in the simulation environment as conglomerates using a set of very abstract "orders" that have no real world analog.

2) Unfortunately there is an inherent incompatibility between the way people exchange information and the way computer simulations can accept and interpret information. Humans use natural language which is rich, but fuzzy. Computer simulations require precise terminology organized in highly structured forms.

The CFOR concept and CCSIL bring a new approach to the construction of simulations that address this problem.

First, by providing an explicit representation of command entities and information exchanges, the CFOR concept provides a more appropriate simulated entity for a human operator to communicate with. A CFOR command entity is busy collecting and reasoning on simulation information that is in a form appropriate for the human operator. The CFOR command entity can reply to requests for tactical state information, as well as, administrative and logistics information.

Second, CCSIL was designed to be both interpretable by software and to be a valid abstraction of the information exchanged by battlefield command entities. The current set of CCSIL messages focuses on providing highly structured, yet flexible formats for the types of information normally conveyed using natural language. The vocabulary of CCSIL messages was selected to coincide with the vocabulary of military personnel. The definitions and semantics for CCSIL vocabulary was originally gleaned from field manuals and is continuously refined to reflect common military usage. Although it is not natural language English, it is much more robust than highly abstract simulation instructions like "Move-Unit" or "Attack".

One piece of ARPA's CFOR program is to use CCSIL and the CFOR infrastructure services software to adapt existing real world command and control systems to interoperate with the simulation components: the CFOR software command entities and the computer generated force representation. This task is described in more detail in the following section.

6. CFOR Software Design

The CFOR architecture is supported by a set of software. The software developed to date has been to support the command entity application developers brought onboard by ARPA to build Army Company Team commanders and other Army commanders in the next 12-18 months. Eventually software will be developed to support command entity development for the other military Services. This work lags 6-12 months behind the Army CFOR work due to the shortfalls in simulations of platforms and small units for the other Services. This paper describes the work done to date.

The CFOR software comprises three components: the adapted C2 workstation application, the infrastructure services software, and the adapted computer generated force application. This section describes the current status of these software components.

6.1 Adapted C2 Workstation

Work is underway to adapt the Army's B2C2 workstation prototype to be compatible with the CFOR simulation environment. Using this Adapted B2C2 Workstation, a commander or staff officer can send and receive CCSIL messages to and from his subordinate units. The Adapted B2C2 Workstation will capitalize on several elements of the Information Services layer of the CFOR architecture.

The intent for the STOW-97 ACTD is to deploy a group of these workstations at Battalion command post mock-ups. Experienced Army personnel or exercise support personnel at these command posts would use these workstations to direct and monitor the activities of the virtual Companies in the virtual simulation environment. They would use these and other real world C2 workstations or communications devices to communicate and exchange data with members of the training audience at brigade, division, other command posts.

As ARPA's Synthetic Forces program continues to extend the type of forces achievable in virtual simulation, new C2 workstations will be added to the family of CFOR applications. For example, in the Navy arena we are looking at the JMCIS workstation as a likely candidate.

6.2 CFOR Infrastructure Services Software

The CFOR infrastructure services software comprises several modules as outlined in the CFOR architecture that provide commonly used functions to the CFOR command entities. This infrastructure consists of layers of software organized into libraries, following the programming practices of ModSAF.

6.2.1 Platform Behaviors Module

Platform Behaviors Services provide a generic interface to a command entity's physical representation on the battlefield. A command entity can be associated with a vehicle or a set of vehicles (e.g., a command post). For example, an Army Company commander

may ride in a tank, a Bradley Fighting Vehicle, a helicopter, or a HMMWV. The Platform Behaviors Services were built to use the basic behaviors implemented in the computer generated force application in the Baseline Infrastructure Layer (e.g., ModSAF in the current application). There are three groups of functions in the Platform Behaviors Services.

Within the Move group, the services allow the command entity to drive his vehicle to a specific location, drive in a specified direction, follow another entity, change the speed his vehicle is traveling, and change the orientation of his vehicle. Within the Sense group, the services allow the command entity to use the full range of sensors on his vehicle to sense other entities around him or to sense distinguishable terrain features around him. Within the Shoot group, the services allow the command entity to fire at a target, to fire at a location, to fire in sector, and to cease firing.

6.2.2 Communications Module

The Communications Module helps a command entity application send and receive CCSIL messages. It offers an application interface to the following CCSIL message utilities:

- Message dispatcher that maintains a queue of incoming messages waiting to be processed.
- Notification mechanism that responds to polling by command entity for new messages.
- Message queue accessor that allows command entities to retrieve incoming messages from the queue.

One new feature that the Communications Module brings to the DIS environment is a capability that insures delivery of the DIS Signal PDU from the sending unit's machine (i.e., CPU) to the receiving units' machines. The capability uses an acknowledgment and retransmission (up to three times) scheme to insure delivery of the Signal PDUs containing CCSIL messages. As the DIS protocol evolves and TCP-IP multicasting services become available, we will remove this feature from the CFOR Communications Module.

Note that this feature of insuring delivery of the Signal PDU is not the same as insuring delivery of the message between two command entities in the simulation. Realistic modeling of real world communications devices is a multi-faceted problem. The Communications Module software provides one piece of the large problem of simulating communications devices. It compares the radio identifier and frequency on incoming messages with the radio identifiers and frequencies to which they are tuned for the units being simulated. If any of the units simulated have radios tuned to that frequency, then the message is passed along to the unit. Otherwise the message is discarded. In this way, simulated units listening to the wrong communications net (i.e., tuned to the wrong frequency) will miss messages broadcast on the net that they were supposed to be listening to.

The remaining aspects of communications effects modeling (e.g., propagation loss due to jamming, geography, and weather) are not provided by the CFOR infrastructure services software. Rather, the Communications Module hands the CCSIL messages over to the radio models in the simulation applications being used. In the current version, ModSAF 1.3 has no simulation of the communications devices (e.g., radios). Solving this aspect of the communications modeling problem is not part of the CFOR program.

6.2.3 Command and Control Utilities Module

Command and Control Utilities are included in the CFOR infrastructure to give command entities access to "routine" knowledge, shared by every human commander, that does not depend on subjective judgments. This is important for several reasons:

- To prevent redundant and potentially inconsistent knowledge acquisition and engineering efforts by the command entity developers.
- To help focus the activities of the command entity developers on addressing the difficult issues in modeling subjective, context-sensitive judgments and decisions.
- To localize, as much as possible, the encoding of doctrinal information within the CFOR family of application software for two reasons: 1) to facilitate CFOR testing and evaluation; and 2) to minimize the effort needed for future enhancements or modifications for particular exercises or scenarios.

This capability is implemented using a collection of software modules that have an input parameter list and return one or more data structures of information. The information is generated using basic data retrieval operations and simple assessment functions. These services have been designed to avoid making any inferences or judgments that should be made by the command entities themselves.

The Command and Control Services are organized into four subject areas: Environment Utilities; Unit Information; Tactics, Techniques, and Procedures; and, Missions and Tasks. The current release of the infrastructure software includes an interface specification for the Environment Utilities along with the software modules implementing that functionality. Complete interface specifications and software implementations for the other C2 services are scheduled for the April 1995 release of the CFOR infrastructure software.

6.2.4 Environmental Utilities

The goal of the Environmental Utilities (EU) is to provide an interface to S1000 terrain data that supports automated decision making. The utilities focus primarily on factors affecting movement of vehicles, cover, and delivery of fires for lower-echelon units.

The S1000 data format, used for SIMNET and ModSAF simulations, is efficient for real-time graphical display of terrain, but does not directly support automated command entity reasoning. The EU interface to this data is presented in a series of layers, as follows.

- LAYER 0 provides implementation-independent access to the basic terrain data: elevation, slope, soil type, feature type and location, platform capabilities, time of day, etc. Provided analyses at this layer include trafficability, line of sight, and coordinate conversion. Most data is floating point.
- LAYER 1 accepts dynamic, user-defined no-go overlays. These specify areas of operation, avenues of approach, area obstacles, and other mutable restrictions to movement. Subsequent analyses (at higher layers) will respect any active overlays.
- LAYER 2a provides a fully connected topology of vertices, edges, and faces atop Layer 0. Faces are guaranteed to be planar and of uniform soil type. Linear edges guarantee constant trafficability for a point platform. Full connectivity allows consistent and convenient movement-oriented terrain reasoning. Analysis at this level includes a general-purpose route finder, weapons fan, and cover determination. Data is integer where suitable.
- LAYER 2b provides a graph topology representing precomputed mobility corridors. These are used to reason about trafficability of aggregate entities that require a doctrinal frontage. A corridor segment guarantees uniform width and trafficability. Analysis at this layer includes identifying the mobility corridor nearest to a query point, and corridor-based routing. Data is integer where suitable.

During an exercise, layers 0, 2a, and 2b are static for a given playbox (for example, the ModSAF 1.2 Fort Knox terrain database).

The EU package is structured to be independent of the underlying terrain representation where possible and to anticipate near-term developments (such as dynamic terrain) by including place holders for these concepts in the design. In this way, the infrastructure can help command entity developers anticipate and design to future needs using existing software.

6.2.5 Unit Information

The purpose of the Unit Information utilities is to assist in providing command entities with part of the minimal body of information expected of all commanders, no matter what their branch, experience, or expertise. These functions fall generally into two categories:

- Static information found in the battle books, field manuals, and technical manuals that commanders have access to during combat and training situations. This includes such details as unit sizes and compositions, weapon and vehicle data, and estimated times to complete certain tasks. For example, a commander may look up the composition of a motorized rifle regiment to determine their composition, then lookup the ranges of the weapons in an Motorized Rifle Regiment to determine their maximum range and effects.
- Unit assessment functions, which aggregate raw data into commonly used terms. For example, when a commander wishes to report the location of his unit, he can obtain his unit location by calling a function with the location of each subunit in his command.

6.2.6 Missions and Tasks

The Missions and Tasks services provide a command entity with a skeletal decomposition of standard Company Team operations into tactically meaningful components, along with guidelines for implementing the tasks and subtasks associated with each component. The rationale for providing such a declarative representation of doctrine is that a command entity competent in executing all the basic components can execute any mission defined in terms of those components. The components for Company teams build on the ARTEP collective tasks. The target repertoire of mission decompositions includes those missions corresponding to the set of virtual training exercises (Attack, Defend, Delay, Movement to Contact, Reconnaissance in Force, Raid, Exploitation, and Pursuit).

Initial implementations will only consider a service that relates missions to tasks. It may be useful to offer other classes of these services that relate tasks to subtasks, or Company tasks to Platoon missions and tasks.

6.2.7 Tactics, Techniques, and Procedures

The Tactics, Techniques and Procedures (TTP) services will provide a command entity with doctrinally acceptable decision options for conducting an operation. These services are designed to present tactical considerations and techniques, standard operating procedures, and "tricks of the trade" in a manner that facilitates the "how to" aspects of a commander's job. The decision options offered typically represent "textbook" solutions that every human commander would recognize from his military education. The motivating rationale for TTP services is to help

command entities in those areas where human commanders can routinely generate an acceptable solution, regardless of their level of competence.

6.3 Adapted ModSAF

Adapted ModSAF is the current CFOR implementation of the Baseline Infrastructure Layer of the architecture. At the time of this paper we are using ModSAF version 1.3. Five libraries have been added to make it CCSIL-compatible. That is, to enhance ModSAF so that ModSAF units receive and react to CCSIL messages from command entities, as well as generate and send CCSIL messages to command entities.

The Communications Module of the infrastructure software can accurately send and receive all CCSIL message types. Unfortunately the adapted ModSAF component cannot respond appropriately to all CCSIL message types. The current release can respond to and react to the following subset of CCSIL messages:

- Operation Order
- Fragmentary Order
- Execute Directive (a subset of the complete family of messages)
- Report-Request

Using these messages, an Armor Company command entity (portrayed by a human in lab tests) can command and control his company through a military scenario that we call a virtual training exercise. A virtual training exercise is a military operation that logically combines a sequence of Army Training and Evaluation Program (ARTEP) tasks in order to test the unit on its ability to execute those tasks.

A CFOR Armor Company (commanded by a human or software command entity), comprising three tank platoons plus the Commander's tank and the Executive Officer's tank, can carry out a subset of the doctrinally prescribed tasks for Tank Platoons (ARTEP 17-237-10-MTP) and Armored/Mechanized Companies (ARTEP 71-1-MTP).[1] ModSAF's tank platoons and armored/mechanized companies can perform the fundamental fire and movement tasks. Tasks (or behaviors) that are more abstract, less visually observable, are not supported as well. For example, "assist passage of lines." Other tasks that are not supported involve interactions with the terrain that are either vaguely supported or not supported at all in the DIS environment. For example, "construct a hasty obstacle" and "execute a prepared obstacle" require the capability to move earth or environmental features (e.g., fell trees) and effect a semi-permanent change to the terrain. As the DIS community finds solutions to these problems, ModSAF units can evolve to perform more doctrinally prescribed tasks. In turn, CFOR command entities can grow to take advantage of fully-capable subordinate units.

The tank platoons in adapted ModSAF have been modified to generate and send the following CCSIL messages:

- Unit Situation Report
- Unit Status Report

This adaptation is transferable to all unit organizations in ModSAF. As ModSAF's capability to represent platforms and units expands over the next fiscal year, the CFOR program's adaptations will expand as well.

7. CFOR Implementation Plan

The program plan for CFOR calls for multiple, concurrent activities:

- *Knowledge Acquisition.* Experts in each field and for each military Service will gather information about the command process. Particular emphasis will be placed on planning, decision-making, monitoring, and revising plans.
- *Infrastructure Implementation.* As described earlier, the CFOR infrastructure will provide services to the command entity simulation and the real world C2 systems used in a simulation exercise. An initial delivery of this software was made in January 1995; new versions will be issued every three to six months.
- *Command Entity Simulation Implementation.* The CFOR program plan calls for multiple contractors, each developing a software implementation of a command entity. After a suitable period of development, the implementations will be evaluated. Subsequently, the developers will deliver new and improved command entities every six months until the 1997 demonstration. It is expected that initial experience will be gained in implementing Army command entities and that experience will be applied to implementing those of the other military services.
- *ModSAF Enhancement.* ModSAF will be enhanced to model new entities (vehicles and small units), to model new behaviors for entities and small units, and to properly carry out CCSIL orders and requests and to generate CCSIL reports.
- *C2 Workstation Adaptation* Beginning with the Brigade and Battalion Command and Control system (B2C2), selected examples of real world C2 systems will be adapted to work in a virtual simula-

[1] Tank platoons in ModSAF version 1.3 can perform 22 of 36 prescribed ARTEP tasks. Armored/mechanized companies in ModSAF version 1.3 can perform 8 of 30 prescribed ARTEP tasks.

tion exercise. The C2 systems will be modified to use the CFOR infrastructure services software to send and receive CCSIL messages and to control the physical portrayal of the commander in the virtual simulation environment (e.g., to move the command post from one location to another).

8. Acknowledgments

This work is being supported by the ARPA Advanced Distributed Simulation Project. COL Robert Reddy, MG Larry Budge (US Army retired), and LCDR Peggy Feldmann, Program Manager for Synthetic Forces, have provided continuous support and guidance. Mr. Kent Pickett and the Eagle development team at TRAC-Ft. Leavenworth provided the groundbreaking work in explicit modeling of command and control in combat simulations and the Battle Management Language. Mr. Ben King, Mr. Kurt Louis, Mr. Jeff Pace, and Mr. James Hughes developed the CFOR infrastructure services software.

9. References

MITRE Corporation, January 1995 *Command and Control Simulation Interface Language (CCSIL) Message Content Definitions*, McLean VA.

MITRE Corporation, January 1995 *Command and Control Simulation Interface Language (CCSIL) Appendix A: Enumerations,* McLean VA.

MITRE Corporation, January 1995 *Command and Control Simulation Interface Language (CCSIL) Appendix B: Formatted Data Types,* McLean VA.

MITRE Corporation, January 1995, *Command and Control Simulation Interface Language (CCSIL) Usage and Guidance*, McLean VA.

MITRE Corporation, January 1995, *Command Forces (CFOR) Environment Utilities Application Programmer's Interface (API)*, McLean VA.

MITRE Corporation, January 1995, *Command Forces (CFOR) Infrastructure Interface Definition*, McLean VA.

Salisbury, Marnie, March 1995, *Command and Control Simulation Interface Language (CCSIL): Status Update,* McLean VA.

10. Authors' Biographies

Marnie R. Salisbury is a Member of the Technical Staff at the MITRE Corporation. She is the task leader of the CFOR CCSIL effort and primary author of CCSIL. Ms. Salisbury has ten years experience in military command and control and battle simulation.

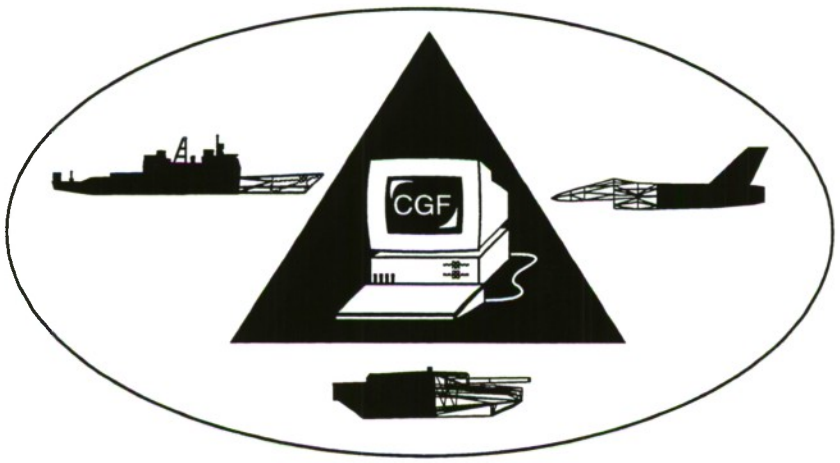
Lashon B. Booker received the Ph.D. in Computer and Communication Sciences from the University of Michigan in 1982. Dr. Booker joined MITRE in August 1990 and is currently a Principal Scientist in the Artificial Intelligence Technical Center. From 1982 to 1990, Dr. Booker worked at the Naval Research Laboratory where he was head of the Intelligent Decision Aids Section in the Navy Center for Applied Research in Artificial Intelligence.

David W. Seidel is project leader for the Command Forces effort at the MITRE Corporation. For the past five years he has supported DoD agencies in integrating military simulations. He has over fifteen years of experience in development and application of military wargames.

Judith S. Dahmann currently leads the MITRE Advanced Simulation Program Area. With twenty years of experience with command and control and simulation technology, Dr. Dahmann has led several efforts to develop and transition these technologies, including the Aggregate Level Simulation Protocol, to effective battlefield application. Dr. Dahmann holds a Masters degree from the University of Chicago and a Doctorate from the Johns Hopkins University.

Session 9a: Experimental Results

Craft, UCF/IST
Karr, UCF/IST
Rajput, UCF/IST
Schricker, UCF/IST


CGF

Experimental Conversion of the IST Computer Generated Forces Simulator from C to Ada

Michael A. Craft and Mikel D. Petty
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
mcraft@ist.ucf.edu
mpetty@ist.ucf.edu

1. Executive Summary

This project developed an Ada equivalent of an existing C Simulator, the latter also developed by IST. The C Simulator, a Computer Generated Forces system with a SIMNET interface, is complex enough to assure a worthwhile experiment. The resulting Ada Simulator is equivalent in every meaningful way to the C version.

The original project time estimate was missed by a wide margin (20 versus 54 person-months) but the experiment was of a much better caliber than originally outlined. Whereas a "conversion" was envisioned, it became clear early in the project that a redesign was essential to take advantage of Ada's capabilities. In particular, the Ada Simulator makes heavy use of Ada tasking, whereas the C version used its own executive: this leaves the Ada version at the mercy of the run time support but gives a much better and more portable environment.

A peculiarity of the PC architecture (real versus protected mode) forced compromises in the networking support. A great deal of effort was devoted to solving this problem and the solution is not completely satisfactory (it may be a time bottleneck).

Compilation time became a real problem as the project matured. A C Simulator can be built in about 5 minutes; the Ada version takes over 2 hours on equivalent machines. Even worse, minor changes often force 20 minute compiles.

The Ada Simulator's performance was uniformly inferior to the C version's. Depending on how the data is examined, one could argue the Ada version is anywhere from 3 to 5 times as slow. The reason for this is not well understood, although the network support is suspect (as is the overhead incurred by our heavy use of tasking).

It is commonly believed that Ada projects are "much larger" than their C equivalents but we did not find this to be so. The Ada Simulator is larger, but only by about 11%.

All the senior members of the team had worked on the C Simulator and knew it well. Nonetheless, all agreed the Ada product was superior in a software engineering sense. Objective measures of complexity bear this out.

2. Introduction

Considerable controversy has surrounded the question of what programming language should be used for developing computer software for simulation systems. The Ada and C (or C++) languages each have their proponents. Many of the language selection discussions are based on comparisons of language features or reports of development projects in one language or the other. A direct comparison of two implementations, one in C and one in Ada, of a simulation software system of significant size has rarely been undertaken because of cost considerations.

IST had previously developed a CGF Simulator as an environment in which to perform CGF research. That research software system is written in ANSI C. In the experiment described herein, a version of that CGF Simulator was re-implemented in Ada so as to provide a direct comparison of C and Ada, at least for the specific purpose of implementing CGF systems.

To perform the experiment, a stable version of the CGF Simulator was frozen for the purposes of the conversion experiment. The team assigned to the experiment then redesigned and re-implemented a functionally equivalent CGF Simulator in Ada (and only Ada, there is no assembly or C in the Ada Simulator). There are two important points to keep in mind about the conversion. First, great care and effort was expended to produce an Ada Simulator that was functionally equivalent to the C Simulator to the extent achievable, so as to make the comparison as direct as possible. Second, the Ada Simulator was a redesign of the Simulator, based on full use of Ada language constructs.

At the conclusion of the conversion, the two CGF Simulators were tested for functional equivalence and run-time performance.

From the experiment we expected to learn which language was better suited for the specific problem of implementing Computer Generated Forces systems for real-time virtual battlefield simulations. We also hoped to gain insight into the larger question of what language to use for general simulation systems. Finally, we wanted to be able to confirm or deny the often heard opinion that "C should be used for research systems and Ada for production systems." The full technical report on this experiment is [Craft, 1994].

3. Project Overview

The Ada project was undertaken by a group of people (the "Ada Team") who were, in most cases, intimately familiar with the C Simulator; a lack of knowledge of the C Simulator was never a critical issue.

The Ada Simulator was to be equivalent in every meaningful way to the C version; it was even hoped the Ada Simulator would pass a sort of Turing Test -- a user of a Simulator would be unable to tell whether the C or Ada version was in use.

The new Simulator was to be evaluated using the C version as a baseline.

It makes sense to base an Ada conversion project of this kind on IST's Simulator. The IST Simulator is sufficiently complex (offering semi-automated behavior for scores of vehicles with interconnection through the SIMNET protocol) and yet sufficiently tractable (it is completely self-contained, right down to its own executive, and yet has less than 60,000 lines of code) to make it an excellent target.

The Ada Team took a "can do" approach to producing an Ada equivalent. The assumption was, contrary anecdotes not withstanding, that an Ada equivalent could be built that was not "too much" larger and, at worst, "slightly" slower than the C version. Further, the Ada Team expected to produce a better overall product.

3.1 The Baseline Simulator

This work was performed in the IST CGF Testbed, an environment for testing CGF behavioral control algorithms developed under the sponsorship of ARPA and STRICOM (Danisas et. al. 1990, Gonzalez et. al. 1990, Petty 1992, Smith et. al. 1992a, and Smith et. al. 1992b).

IST's 6.4 Simulator, which is the basis of this project, is the 4th release of the Simulator's 6th major revision; it is a mature product. The C Simulator had been subjected, many times, to rigorous evaluations leading to large scale refinements both in performance and reliability.

3.1.1 Key Capabilities

The Simulator supports semi-automated forces, including infantry, suitable for use as opponents in training scenarios intended for human participants.

Action is displayed through a graphic "bird's eye view" called the Plan View Display (PVD). The user can control the scale of the display and what area of the terrain is shown. The display shows terrain features as well as vehicles and infantry.

A user can create as many as 12 entities on a Simulator, selected from over 20 entity types. These can be made to carry out a variety of activities including planning routes, following routes (planned or supplied), and opening fire (selecting and firing appropriate weapons chosen from scores of munition types).

Remote entities are recognized and shown on the PVD. The Simulator interacts with remote vehicles

(e.g., local infantry can get on board (mount) remote vehicles).

The entities created have sensible approximations of correct behavior: they accelerate and travel at feasible rates, they compute line of sight (LOS) and "understand" what they see (a key point in target selection), and the entities use appropriate weapons. There are many other such considerations carried out by the Simulator.

The Simulator is driven by 2 external sources, the keyboard and the network, each representing at least 2 types of control.

Human beings enter Simulator commands through the keyboard. The keyboard is logically extended by the use of script files, which give keyboard commands through a file, rather than actually striking keys.

At least two types of network traffic are handled. The obvious traffic is of the SIMNET protocol type, which allows the Simulator to "see" remote vehicles (and to tell the world about its own vehicles). The other type of traffic follows the IST CGF Testbed's "message" protocol. This protocol allows one Simulator to control another Simulator and, more importantly, allows a user-friendly Operator Interface (OI) to control several Simulators.

4. Personnel

It is unfortunate that the personnel roster was somewhat unstable, but this is not atypical of such a project, especially when student assistants are employed. The staff instability cannot be ignored as a factor in the project's duration, so a chart indicating the staff status is supplied here. The initials on the vertical axis identify specific individuals. Split blocks indicate part time employees.

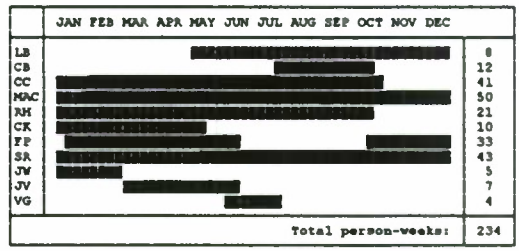


Table 1: Personnel Roster

The total time expanded was approximately 54 person months (170% more than was originally estimated.

4.1 Original Project Estimates

The project, including evaluations and technical report production, was expected to take less than 6 calendar months (the project was first estimated at 20 person-months [Petty, 1991]).

The original estimate discussed things which, in retrospect, were ridiculous considerations, such as how to translate comments and constants. To be fair, when that was written the translation was anticipated to be much closer to the C Simulator than was to prove to be the case.

The estimate explicitly stated the project was to yield a "language conversion, not a software redesign." A particularly telling statement in the estimate indicated Ada tasking would not play a role in the new Simulator.

Time and again the Ada Team members agreed that the decision to use Ada tasking was correct; some felt it is Ada tasking that, more than any other feature, showed Ada to be the right choice for Simulator development. Having such a capability in the language greatly simplifies the designer's work and greatly increases portability.

While it is true the original estimate was missed by a wide margin (running over 170%), the experiment was of a much higher caliber than the original goal outlined. The deviation from what was intended is largely because of the seductive nature of the Ada language.

5. Technical Problems

A project as complex as this severely tests the development tools used (hardware and software). Difficult technical problems must be expected. The major problems for this project were in the areas of the PC's Real and Protected modes and compiler limitations.

5.1 Real-mode Versus Protected Mode

In brief, the IBM PC and compatibles can operate in two modes, "protected" and "real." DOS runs in real mode, which has its roots in the earliest PC architectures (the 8086 family).

The Alsys compiler produces protected mode code, whereas the C Simulator is a real mode program. However, the packet driver, which both Simulators use to send and receive network packets, is a real mode program, and so a protected/real mode interface must stand between the Ada Simulator and the packet driver (IST did not have access to, and it was impractical to build, a protected mode packet driver). For packets being transmitted this is straightforward, but handling incoming packets is problematic.

Software to support these interactions is defined by the DOS Protected Mode Interface (DPMI) [INTEL, 1993]. Since the DPMI specifies what the Ada Team thought was needed, no particular difficulty was anticipated once the problem was understood.

Unfortunately, the interfaces available to IST did not implement a feature of the DPMI which was needed (handling "call backs").

IST avoided the problem by writing another real mode program to field the packet interrupts and queue incoming traffic. This new program is in the form of a Terminate and Stay Resident (TSR) program, and so remains active and accessible once loaded. The Ada Simulator gets packets by polling this new program.

Polling (in general) is not desirable, but no alternative was found. The Ada Simulator only polls when no other activity is scheduled which tends to make the Ada Simulator show stress through lost packets before entity behavior breaks down.

5.2 Ada Compiler Limitations

On the whole, the Ada Team was satisfied with the Alsys Ada Compiler. Most of the other Alsys supplied tools (in particular, AdaMake) appeared to have problems and were abandoned quite early in the project.

5.2.1 Compilation Time

The Ada project was scheduled to be done on 33 MHz. 386 PCs with 4 megabytes of memory. As soon as the compilers arrived it was clear the project could not continue without more memory. Within 3 months it was necessary to switch to 486 based PCs to continue. In spite of this, a system build took about an hour 4 months into the project (for 20,000 lines of code).

After 6 months system compilation time was over 1 hour 35 minutes and it was obvious the Ada Team would have to eventually reduce disk caches to the point where compilation would be impractical. Additional memory was installed bringing every machine up to at least 16 megabytes. This allowed disk caches to be expanded and was a great help. Nonetheless, a great deal of our development work involved "simple" changes which yielded a specific chain of compilations which take about 20 minutes.

At each stage of the project, system reorganizations were made to reduce file dependencies and so compilation times

Experienced Ada professionals have shown no surprise at the length of the compilations. The Ada Team, however, which was accustomed to Borland C++ compilers building the C Simulator in about 5 minutes, found the long compilations a constant and serious irritant.

The long compilation time affected the less experienced programmers the most. Some of the part time student programmers never seemed able to understand how to get the most out of a compilation and their productivity suffered as a consequence.

At the project close a complete system build required a minimum of 2 hours 16 minutes.

5.2.2 Compiler Bugs

There were serious problems with the Alsys compiler support tools. AdaMake, for example, which was supposed to help Ada programmers compile only what was necessary never worked properly. Borland's "make" utility was used in conjunction with an IST developed dependency generator (creating it was a non-trivial task).

The compiler itself showed some intermittent bugs and one serious, consistent, bug.

The intermittent bugs were manifested as compiler crashes during compilations. In the history of the project (which probably generated at least 100,000 compilations) there were only about a dozen of these unexplained crashes. These crashes appeared to be caused by, or to cause, library corruption and so often set work back several hours.

A serious bug appeared when a certain generic was instantiated. The compiler would abort with an internal error while compiling the file that contained the instantiation. The problem was eventually avoided by introducing dummy packages.

6. Results

From the outset it was expected the Ada version would be better in terms of software engineering (human) issues but the C version would exhibit better performance. The experiment confirms these expectations.

6.1 Run-time Performance

Nobody on the Ada project expected the Ada Simulator to out perform the C version. C is a system implementation language; it allows virtually any desired optimization. Further, the C Simulator has undergone many revisions over the years intended to improve performance.

The protocol code was written by the same person (Craft) for both the C and Ada versions. Some of the techniques used in the C rendition were deemed inappropriate in an Ada design and were not carried forward. Since the protocol author had optimized the C version as best as he was able, it was unlikely he would be able to get as good performance in the Ada version since the work was done in Ada with a reduced tool set. It should be noted, however, that the Ada protocol code is, as a consequence of the different techniques used, much more straightforward. Further, the Ada code, because of Ada's exception handling capability, is much more robust than the C version.

6.1.1 Performance Testing Methodology

Two questions must be answered to evaluate performance: what objective conditions will be used to indicate a lack of performance and what parameters will be adjusted to illicit a breakdown in performance?

To recognize stress (lack of performance) objectively, a vehicle was told to route in such a way as to cause it to collide with another vehicle (which is driving in a straight line). Under sufficient stress, the System Under Test (SUT) breaks down in such a way that the vehicles do not collide, or, if network traffic is sufficient it will discard packets. If the vehicles do not collide as expected or if more than 1% of packets are discarded, the stress threshold is considered to be passed (i.e., the SUT fails the test).

There are two fundamental types of stress which bear on the Simulator: internal stress and external stress.

6.1.1.1 External Stress

External stress comes about from attempts to interact with the world outside of the Simulator's code and internal data. This includes interaction with files (reading a terrain database), interaction with a human user (through the keyboard), and interaction with remote Simulators through the network. Only the third item was studied; it is universally recognized that network traffic can cause Simulator problems and tests for network stress are manageable.

To avoid logistics problems in designing and executing repeatable tests to be applied to the C and Ada Simulators, a tool was developed to generate traffic from many vehicles at a controllable rate. All the network stress tests were done using this tool.

6.1.1.2 Internal Stress

Internal stress comes about as the Simulator is required to carry out more work to maintain its vehicles' representations and behaviors.

Both Simulators can support 12 vehicles without stress (with their usual configurations). All tests were done with 12 vehicles.

An important and expensive computation done for each local vehicle is a LOS computation (used to answer the question "can I see you?"). Being computationally expensive and easily controlled (the delay between updates is read from a configuration file) it is a natural parameter for stress experiments.

6.1.2 Performance Analysis

Figure 1 captures the essence of the performance results.

On the horizontal axis the Line of Sight rate is shown, on the vertical axis the network load is shown. Points indicate the boundary between acceptable behavior and manifestations of stress.

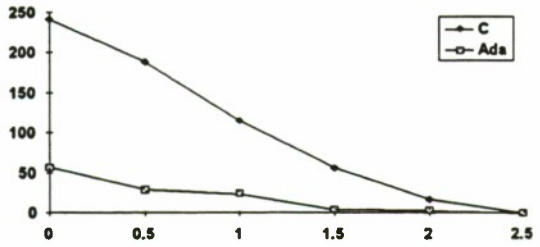


Figure 1: Performance Thresholds for the Ada and C

6.1.2.1 Optimizations

Because of the relatively poor performance of the Ada Simulator relative to the C Simulator, methods to improve the Ada Simulator's performance were studied.

The Ada Team tried several methods: some of the various checks generated by the compiler were suppressed; direct rendezvous replaced some instances of daisy chained message passing; and the executive's priority was adjusted. The latter idea was recognized as dangerous and unlikely to help.

None of these, individually or in combination, had any effect out of the bounds of experimental error except when there was no network traffic.

The effect of Ada constraint checks may often be overstated, as (for example) they need only be done when a type change is taking place. Nonetheless, the checks do have some impact, so it was surprising to see no improvement. The checks were shut off in what were believed to be the most important areas.

Shutting off the checks in protocol code is not practical as constraint errors are the basis for filtering suspect packets. Even with this extreme approach, no significant improvement was noted.

6.2 Software Quality

As is pointed out by [Sommerville, 1992], the state of software science is such that we cannot directly measure software quality. Rather, we develop objective measurements for things we believe to be related to the quality such as the program's size or the number of conditions in the program. These sections describe what was measured, how the measurements were done, and the results of the measurements.

6.2.1 Relative Sizes

The C sources consist of 58,185 lines; 32,373 of those lines are neither blank nor comments. The Ada sources consist of 64,233 lines, 45,288 lines of which are neither blank nor comments. It may be worth noting that the Ada version consists of about 30% comments and blank lines, while the C version consists of about 44% blanks and comments. This may simply indicate the C version is better documented (it is, after all, a much more mature product) or it may be a sign that the C version requires more documentation to be understandable; it is probably a combination of these and other factors.

Based on these figures, the complete Ada sources are 11% larger than the C sources, and the non-blank, non-comment Ada sources are 40% larger than their C counterparts.

The nature of Ada is such that people have come to expect larger sources than when C is used (C is very compact, using combinations of ++, ?:, loops whose exit conditions contain more than tests, and the like, whereas Ada is verbose, requiring end-if and end-loop, null, etc.). Further, the Ada design incorporates specific data types for almost every contingency and this leads to larger source files.

The Ada Simulator sources contain 314 globally visible functions and procedures, whereas the C

Simulator contains 432 functions and procedures (this does not count macros).

The Ada sources consist of 249 files, the C Simulator sources are made up of 138 "cpp" files and 78 "include" files, for a total of 216 files.

The Ada sources are divided among 4 directories, with 176 package specifications. The C sources are divided among 52 directories.

To give a graphical comparison of the two Simulator's sources, figure 2 shows the relative concentration of (all) lines, lines of code, functions and procedures, and files between the two programs. For example, 52% of the lines making up both Simulators is found in the Ada version.

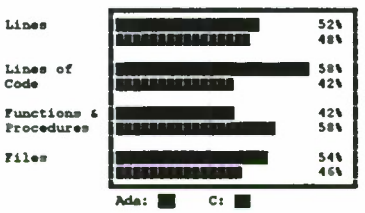


Figure 2: Relative Program Sizes

6.2.2 Complexity

To measure software complexity a software tool was developed to count the number of decisions made. This metric has been in use for years [McCabe, 1976] and experience has shown it to be a reasonable indication of the human perception of a program's complexity.

Consider two programs of equal length, one with no conditions (and so no indentation, just statement after statement is executed, always in sequence) while the other program has several tests, possibly imbedded. Even if the programs are of equal length, most people agree the latter is more "complex."

As a plausibility check on the tool developed, and the method in general, it was used to select the "most complex" of the Ada files and the "most complex" of the C files. Before knowing the results of the test, a subjective selection for the most complex file was made by a senior team member.

Although the top Ada selection was a surprise, after examining it was agreed that it was, indeed, very complex. The tool's second choice matched the team member's choice for first place. The top choice for the C Simulator is complex but a pathological case, the second choice matched the team's selection.

6.2.2.1 Complexity Results

It is disappointing to find the Ada tests are denser than the C tests, this was not anticipated.

The C version contains 3,075 tests in 32,373 lines of code (blanks and comments ignored). This means there are 9.5 tests per 100 lines of C. The Ada version contains 4,690 tests in 45,288 lines of code (again, blanks and comments are ignored). This means there are 10.4 tests per 100 lines of Ada.

The C version uses extensive headers (essentially test free), many supplied by outside agencies containing hundreds of lines which are little more than filler for the purposes of this measurement. To see if this plays an important part, the calculations were re-done excluding the C headers and excluding the Ada files which contain nothing but package specifications.

With these adjustments, the C version has 3,052 tests out of 23,341 lines, or 13.1 tests out of 100 lines. The corresponding Ada version has 4,625 tests in 36,574 lines, or 12.7 tests per 100 lines. Further analysis along these lines (removing any remaining specifications, removing "#define" from the C sources) would be difficult and questionable.

It appears the C and Ada Simulator sources are of about the same complexity using the described metric.

6.2.3 Human Factors

The Ada Team members are familiar with, and have worked on, both Simulators. All the remaining members of the Team, and most of those who left before the project end, have expressed the opinion that the Ada Simulator is easier to understand. One member, who left the project and later returned, expressed the belief that the Ada Simulator was easier to follow.

6.3 Reliability

There is no way to objectively determine reliability for a product as complex as the Ada Simulator. At

best metrics are selected which should indicate the overall health of the target software.

6.3.1 Reliability Metrics

A common hypothesis [Sommerville, 1992] is that the number of "bugs" in software is related (possibly proportional) to the number found in testing if the testing is done well.

The Ada Team attempted to find the relative frequency of errors in the two Simulators by applying a set of identical tests to each.

6.3.2 Method Used to Locate Problems

Tests were designed and applied to each Simulator and the results recorded. Most team members contributed experiments to be run, usually in the form of scripts to be processed.

6.3.3 Verification Results

Of 58 tests the C Simulator failed 19 tests and the Ada Simulator failed 24 tests (failures are common because these scripts were designed to make the Simulators fail). In 4 tests the Ada failures were minor; the problems are understood and could be solved with just a little effort. One C failure was so severe that it caused the C Simulator to crash and required the PC to be re-booted.

Based on these raw statistics, it appears the Ada Simulator is about 27% more error prone than the C version. As already noted, however, 4 of the Ada failures are of a minor nature and would have been fixed had the Ada version undergone any serious testing before this. Excluding these we get rough parity between the Simulators.

The C Simulator is quite mature and has undergone years of debugging. In contrast, the Ada Simulator was brand new with no history of error removal. The small difference in error frequency between the C and Ada Simulators suggests that a similarly mature Ada Simulator would have an advantage. The Ada project spent no time seeking bugs; those uncovered and repaired were found by happenstance during development. With some effort the Ada Team believes the Ada Simulator would improve a great deal and easily surpass the C Simulator's reliability.

7. Conclusions

Ada versions of programs do not have to be very large by comparison with their C counterparts. IST has produced an Ada equivalent of a complex and sophisticated software product that is only 11% larger than its C equivalent (40% larger if blank lines and comments are ignored).

Another important realization is that it is difficult to objectively determine whether one software approach is superior to another. Not a single point of evaluation was as clear cut as was hoped. It is disappointing to have to continue to say that the Ada Team, as a whole, believes the Ada Simulator is the "better" product, but a clear demonstration of this eludes us.

The complexity measure favors the Ada version but only after excluding "include files" from the C version and their (rough) equivalent, package specifications, from the Ada version. This exclusion does seem reasonable.

The Ada performance is uniformly inferior to the C Simulator's performance. It is important to realize the reason for this is not understood: various attempts to improve performance were ineffective. The network interface is considered a likely source of problems but there is no solid objective evidence of this.

People familiar with both systems (i.e., the Ada Team members) think the Ada Simulator is a better product.

7.1 Key Questions

Now that IST has built a Simulator in Ada based directly on a well understood and sufficiently complex C Simulator, it is possible to take a stand on some of the most common questions regarding Ada projects.

- *Is the resulting code too large?*

  No. The Ada equivalent is only about 10% larger than the C version (40% if blanks and comments are excluded). This hardly seems unreasonable if the gains in maintainability and understandability are

granted. If an Ada project "balloons" there may be a design flaw.

- *Is the resulting code too slow?*

  Possibly. There is no escaping the fact that the Ada Simulator cannot handle nearly as much traffic as its C equivalent. Because of the protected mode/real mode dilemma it is quite possible that the Ada Simulator is being unfairly penalized. To determine this Ada drivers for the protocol board need to be written. In the meantime, this remains an open question. The quality of the compiler is also an important issue here. However, contrary to popular belief Ada's constraint checking does not appear to be a major factor.

- *Is the development process too cumbersome?*

  Possibly. The target Simulator is a relatively small project and yet we found ourselves suffering with, what seemed to us, very long compilations. Eventually some compromises were made to speed up compilations. Nonetheless, it does not appear there will be a "compilation explosion," once we passed a certain size we were able to keep the compilation times under control and, in some cases, reduced compilation time. Better tools may reduce development problems.

- *Is the lack of pointers to functions a major handicap?*

  No. At the time a function pointer is declared the profile (signature) of the function is specified. Since the profiles are consistent it is an easy matter to replace the function pointer with an enumeration variable to be used in conjunction with a call dispatcher. Instead of calling the function directly, the dispatcher is called with the same parameters as the target function except that it also takes an enumeration value to select the desired function. There are reasonable answers to the objections sometimes made to this approach.

8. References

Craft, M. A., and Petty, M. D. (1994). "Experimental Conversion of the IST Computer Generated Forces Simulator from C to Ada", Technical Report IST-TR-94-13, Institute for Simulation and Training, April 20 1994.

Danisas, K., Smith, S. H., and Wood, D. D. (1990). "Sequencer/Executive for Modular Simulator Design", Technical Report IST-TR-90-1, Institute for Simulation and Training, University of Central Florida, 16 pages.

Gonzalez, G., Mullally, D. E., Smith, S. H., Vanzant-Hodge, A. F., Watkins, J. E., and Wood, D. D. (1990). "A Testbed for Automated Entity Generation in Distributed Interactive Simulation", Technical Report IST-TR-90-15, Institute for Simulation and Training, University of Central Florida, 37 pages.

INTEL, (1993) "An Introduction to the DOS Protected Mode Interface", Intel Corp, April 1993, 20 pages.

McCabe, T.J. (1976). "A complexity measure". IEEE Transactions Software Engineering, 2 (4), pages 308-320.

Petty, M. D. (1991). "Intelligent Autonomous Behavior by Semi-Automated Forces in Distributed Interactive Simulation", Technical Report IST-PR-91-12, Institute for Simulation and Training, November 15 1991.

Petty, M. D. (1992). "Computer Generated Forces in Battlefield Simulation", Proceedings of the Southeastern Simulation Conference 1992, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 56-71.

Petty & Smith (1991). "Intelligent Autonomous Behavior by Semi-Automated Forces in Distributed Interactive Simulation", A proposal submitted to PM TRADE in response to BAA NTSC 91-02 by the Institute for Simulation and Training.

Smith, S. H., Karr, C. R., Petty, M. D., Franceschini R. W., and Watkins, J. E. (1992a). "The IST Computer Generated Forces Testbed", Technical Report IST-TR-92-7, Institute for Simulation and Training, University of Central Florida.

Smith, S. H., and Petty, M. D. (1992b). "Controlling Autonomous Behavior in Real-Time Simulation", Proceedings of the Southeastern Simulation

Conference 1992, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 27-40.

Sommerville, Ian (1992). "Software Engineering, Fourth Edition". Addison-Wesley Publishing, Reading, Massachusetts.

9. Authors' Biographies

Michael A. Craft is a Research Associate working for the Intelligent Simulated Forces project at the Institute for Simulation and Training. He has a Master of Science degree in Mathematics and a Master of Science in Computer Science. His research interests are in the areas of Software Engineering, Real Time Systems, Simulation, and Network Protocols. Mr. Craft has a broad and diverse background ranging from Office Automation to Missile Tracking Systems.

Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He currently manages the Plowshares emergency management simulation project; previously he led Computer Generated Forces research at IST. Mr. Petty has a B.S. in Computer Science from the California State University Sacramento and a M.S. in Computer Science from the University of Central Florida. His research interests are in simulation and artificial intelligence.

Comparison of the A* and Iterative Deepening A* Graph Search Techniques

Clark R. Karr, Sumeet Rajput, and Larry J. Breneman
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
ckarr@ist.ucf.edu

1. Abstract

A* is a time-efficient search algorithm but shows exponential growth in memory usage. Iterative Deepening A* (IDA*) has been shown to be space-efficient in searching *trees*. This paper compares the two algorithms in terms of run time and memory usage in graph search within a route planning algorithm. This research shows IDA* to be space-efficient in searching *graphs* as well, but the run times are very high. To reduce run times, we propose optimizations to the algorithm that allow IDA* to approach A*'s run times, while using less memory.

2. Introduction

A* is a time-efficient search algorithm and has been used for route planning in Computer Generated Forces (CGF) systems (Karr 1995) (Loral 1994). Because A* shows exponential growth in memory usage, Campbell (1995) use a variant of A*, Iterative Deepening A* (Korf 1985), that has linear growth in memory usage.

Korf (1985) shows Iterative Deepening A* (IDA*) to be space efficient in searching *trees*. It is unclear how IDA* would perform in a graph search. The motivation for this study was to determine the time and space efficiency trade offs for graph search within the CGF route planning domain.

In the Route Planner (RP) described in Karr (1995), a two-dimensional grid is overlaid over the terrain. The grid is divided into a number of grid cells and is populated with obstacles (canopies, treelines, rivers, and lakes). The grid cells contain *Sample Points* which can be reached from other grid cells. These Sample Points are treated as nodes in a graph and a search of the graph finds routes. The RP uses the A* algorithm (Winston 1992) as the search algorithm. For this study, a version of the RP was built that used the IDA* algorithm in place of A*. The two algorithms could then be directly compared.

3. Algorithm descriptions

The following sections present brief descriptions of the A* and IDA* algorithms.

3.1 A* search algorithm

The A* algorithm is a *branch-and-bound* search, with an estimate of the remaining cost, combined with the *dynamic-programming* principle, (Winston 1992). When the estimate of the remaining cost is a lower-bound on the actual cost, A* produces optimal solutions. This estimate of the remaining cost is called the "underestimate". Route cost is the sum of the cost of the partial route and the underestimate to complete the route. A natural underestimate in route planning is the "cheapest" cost of a linear route from the last point on a partial route to the destination. The dynamic programming principle improves search efficiency by retaining only the best partial solution to each point for further analysis.

The following pseudo-code describes the A* procedure (Winston 1992).

Algorithm A*

1. Form a one-element queue consisting of a zero-length route that contains only the root node.

2. Until the first route in the queue terminates at the goal node or the queue is empty,
   - 2.1 Remove the first route from the queue; create new routes by extending this route to the neighbors of the terminal node.
   - 2.2 Reject all new routes with loops.
   - 2.3 If two routes reach a common node, retain only the route with the minimum cost.
   - 2.4 Insert each new route into the queue in ascending cost order.

3. If the goal node is found, announce success; otherwise, announce failure.

3.2 Iterative Deepening A* (IDA*)

IDA* works by doing a depth-first search (DFS) from the initial state until the total cost of the current branch being expanded exceeds a given threshold. If the goal is not reached, the threshold is increased and the DFS is repeated with the new threshold. The threshold starts at the underestimate from the start and the destination. After each iteration the new threshold is set to the minimum cost of all the values that exceeded the previous threshold.

IDA* uses less memory because at any given point in the DFS, only the stack of states corresponding to the current search path is stored. The pseudo-code description of the IDA*, as it was used in the RP, is:

Algorithm IDA*

The Main Loop

1. Initialize threshold to the linear distance between the start and the destination.

2. Until a route is found,
   2.1 Do IDA* (call *Procedure IDA**) search from the start.
   2.2 Update the threshold for the next iteration.

Procedure IDA*

1. If the route being expanded reaches the goal node announce success; otherwise,

2. Determine the locations that are reachable from the terminal node.
   2.1 Expand the route to the next unvisited reachable location. When all are visited, exit.
   2.2 If the cost of the expanded route is greater than the threshold, reject this route; otherwise,
   2.3 Call IDA* recursively.
   2.4 Go to 2.1.

We start by initializing the threshold to the linear distance between the start and the destination (the underestimate). Then, until a route is found, the threshold is advanced and IDA* is called starting at the start. As the threshold increases between iterations, the algorithm can be seen progressing in "waves".

The IDA* procedure is recursive. The route is expanded to each of the reachable locations from the last route point. If the cost of expanding the route is below the threshold, we continue the DFS from the end of the route. If the cost of the expanded route exceeds the threshold, the route is rejected and IDA* backs up. Whenever the threshold is exceeded, the cost of the route is a candidate for the next iteration's threshold. The minimum route cost that exceeds the threshold becomes the next iteration's threshold. The expansion of all nodes up to the current threshold constitutes an *iteration*.

The number of reachable neighbors from a location is the Branching Factor (BF).

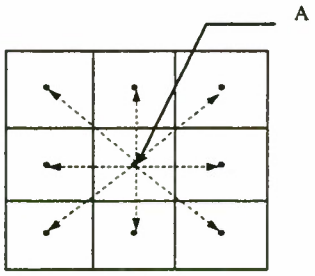


Figure 1: A location with a Branching Factor of eight

For example, in Figure 1, location A's BF is eight because eight locations in neighboring grid cells are reachable. Korf (1985) gives an equation that states the relationship of average BF (i.e., the average of the BFs of all nodes that were expanded in the search) to the run time of the algorithm. The equation reveals that IDA* performance relative to A* improves as the BF increases. Because the maximum BF in the RP is 24, this result provided an additional motivation to compare A* and IDA*.

4. Experimental setup

In the RP, route planning involves searching a grid based terrain abstraction for an optimal route (Karr 1995).

Two factors can affect the performance of the search algorithms.

- Abstract Obstacle Density in the grid and
- Grid cell size.

4.1 Abstract Obstacle Density

Abstract Obstacle Density (AOD) is a measure of the number of abstract obstacles present inside the grid. The terrain is characterized as: Open, Mixed, or Closed. Closed terrain has the highest AOD. Open terrain contains no or very few obstacles and its AOD is nearly zero. Mixed terrain lies between Closed and Open. The AOD was determined subjectively.

4.2 Grid cell size

In Section 5.2.2, we discuss the affects of increasing the threshold between iterations by a fraction of the grid cell size.

4.3 Classification of scenarios

Eight scenarios (A through H) were designed for testing the algorithms' performance. Depending on the AOD of the grid and the grid cell size the eight scenarios were developed:

Terrain type	Grid cell size (meters)				
	5	85	300	530	550
Open		A			
Mixed	F	G	B		C
Closed		D	E	H	

Table 1: Classification of scenarios

Each scenario represents conditions for routing in a specific type of terrain using a specific grid cell size. For example, scenario B utilizes a grid lying on Mixed terrain with the grid cell size being 300 meters.

5. Results

For each algorithm, we compute the time to find a route and the total memory utilized. All the experimental runs were done once; no run to run variability was expected.

5.1 Comparison of A* and IDA*

The run times of the two algorithms are shown in Figure 2.

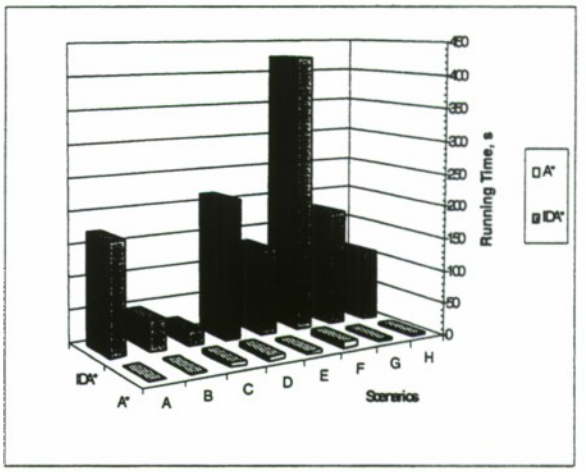


Figure 2: Run times (seconds) of A* and IDA*

For all scenarios, IDA* takes substantially more time than A* in determining the route. For some cases, e.g., scenario F, the difference is very large. Table 2 summarizes the run times of A* and IDA*.

Algorithm	Best	Worst	Average
A*	1.10	9.62	4.58
IDA*	22.09	424.78	161.55

Table 2: Run times (seconds) of A* and IDA*

On average, A* is 35.27 times faster than IDA*.

The memory usage of the two algorithms is shown in Figure 3. Note that the bars showing A*'s and IDA*'s memory usage have been flipped with respect to the bars in Figure 2.

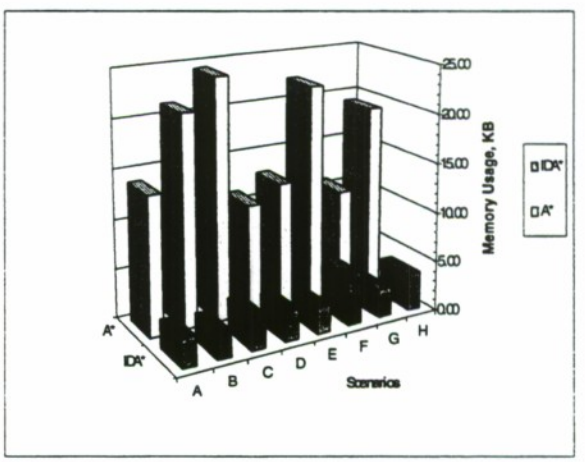


Figure 3: Memory usage (Kb) of A* and IDA*

For all scenarios, A* uses substantially more memory than IDA* in determining the route. For some cases, e.g., scenario C, the difference is very large. Table 3 summarizes the memory usage of A* and IDA*.

Algorithm	Best	Worst	Average
A*	10.77	24.34	16.92
IDA*	2.59	5.11	3.25

Table 3: Memory usage (Kb) of A* and IDA*

On average, A* consumes 5.21 times as much memory as IDA*.

These two results show that even though IDA* is more space efficient, it is less time efficient than A*. The space-time tradeoff, mentioned in Computer Science, is seen here.

5.2 Optimizing IDA*

By using additional memory, IDA* can be optimized to yield run times that approach A*'s. The following sections discuss the optimizations that were done to speed up IDA*.

5.2.1 Storing reachable locations in the grid cell

In a graph, determining a node's neighbors is an O(n) time operation. When neighbors are needed, an Adjacency Matrix (Even 1979) type structure is usually consulted. In the RP, the neighbors of each cell are stored but the reachable locations within neighboring cells are computed during the search. A* determines a node's neighbors once as a consequence of the dynamic programming principle (Winston 1992). Because IDA* makes several iterations over parts of the grid, recomputing reachable locations during the search penalizes IDA*.

In order to establish a fair comparison with A*, the IDA* procedure was modified so that when a node is expanded its reachable locations are stored as part of the grid cell structure. In subsequent iterations, this information is available and the time to recompute the reachable locations is saved.

The run times and memory usage of A* and the optimized IDA* version that stores reachable locations in the grid cell, designated IDA*p, are shown in Figures 4 and 5.

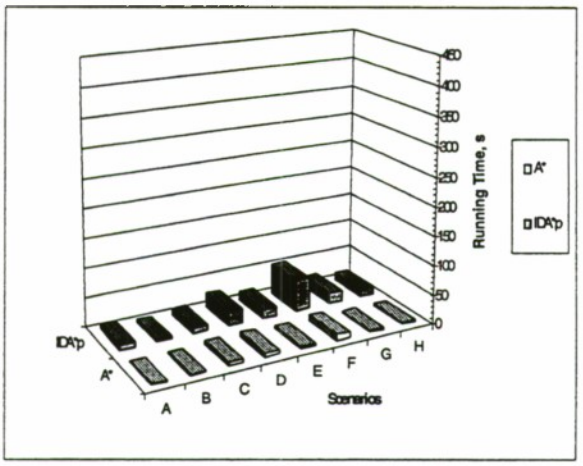


Figure 4: Run times (seconds) of A* and IDA*p

The results reveal a substantial drop in the run times of IDA*. Table 4 summarizes the run times of A* and IDA*p.

Algorithm	Best	Worst	Average
A*	1.10	9.62	4.58
IDA*p	4.51	53.08	18.55

Table 4: Run times (seconds) of A* and IDA*p

On average, A* is 4.05 times faster than IDA*p.

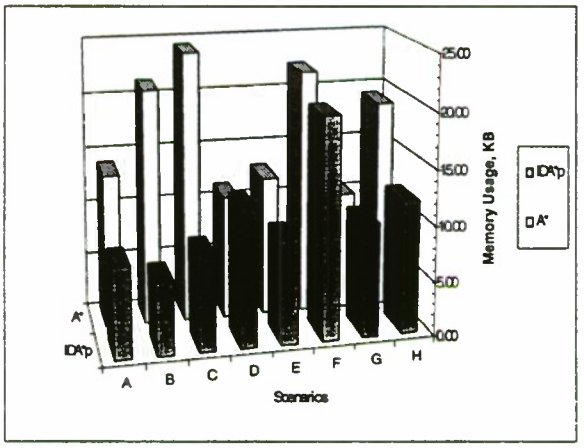


Figure 5: Memory usage (Kb) of A* and IDA*p

As expected, the memory consumption of IDA* has increased because reachable locations are stored as part of the grid cell. However, for most of the scenarios the memory usage is still below that of A*'s. Table 5 summarizes the memory usage of A* and IDA*p.

Algorithm	Best	Worst	Average
A*	10.77	24.34	16.92
IDA*p	6.58	19.96	10.94

Table 5: Memory usage (Kb) of A* and IDA*p

On average, A* consumes 1.55 times as much memory as IDA*p.

It is not possible to predict the additional memory consumption caused by using IDA*p for any scenario. In the worst case, the entire grid would be searched and the memory for reachable locations for all cells would be an upper bound. However, this is likely to be considerably less than A*'s worst case memory requirements.

5.2.2 Using larger thresholds

After each IDA* iteration, the threshold is set to a new threshold (Section 3.2). Normally, the new threshold is the minimum value of the route costs that exceeded the previous threshold. The threshold increment was observed to generally be small (1-2 meters). To test whether a larger increment improved performance, IDA* was modified to use a user defined *advancement_increment*.

If the difference between the previous and new thresholds is less than the *advancement_increment*, the new threshold is set to the sum of the previous threshold and the *advancement_increment*.

Figure 6 shows the run times of IDA* using different *advancement_increments*. The *p* in the notation *IDA*p, t=X* means that IDA* stores reachability information to neighbors in the grid cell (Section 5.2.1). The *t=X* means that *advancement_increment* is set to *X* times the grid cell size.

The shape of the run time curves indicate that as soon as user-defined threshold increments are introduced, run times decrease, and then increase with the increments; i.e., the curves "dip" initially and then rise. The best run times are seen at the "dip". This may seem counter-intuitive as one would expect the algorithm to compute a route faster when larger *advancement_increments* are used.

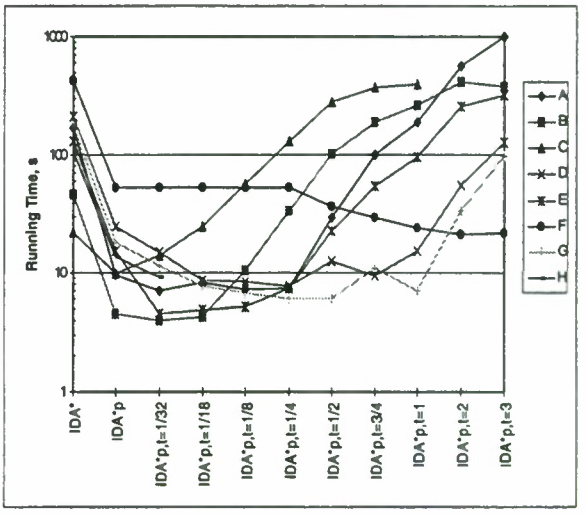


Figure 6: Run times (seconds) of IDA* with different thresholds

IDA*p, t=1/32 shows the fastest run times among different IDA*p, t=X versions for four scenarios (A, B, C, and E). IDA*p, t=1/4 had the fastest run times on two scenarios (D and G). The fastest run time for scenario F occurred at t = 2 apparently due to the small (5 meter) grid size. IDA*p, t=1/32 showed the fastest run time over all scenarios. Figure 7 shows the run times of A* and IDA*p, t=1/32.

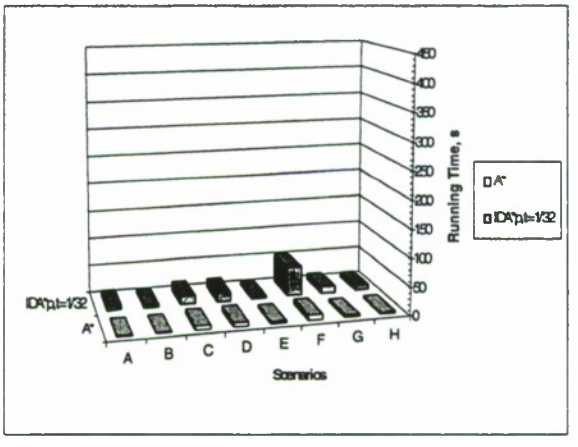


Figure 7: Run times (seconds) of A* and IDA*p, t=1/32

Table 6 summarizes the run times of A* and IDA*p, t=1/32.

Algorithm	Best	Worst	Average
A*	1.10	9.62	4.58
IDA*p, t=1/32	4.01	52.80	14.76

Table 6: Run times (seconds) of A* and IDA*p, t=1/32

On average A* is 3.22 times faster than IDA*p, t=1/32.

Figure 8 shows the memory usage of IDA* using different *advancement_increments*.

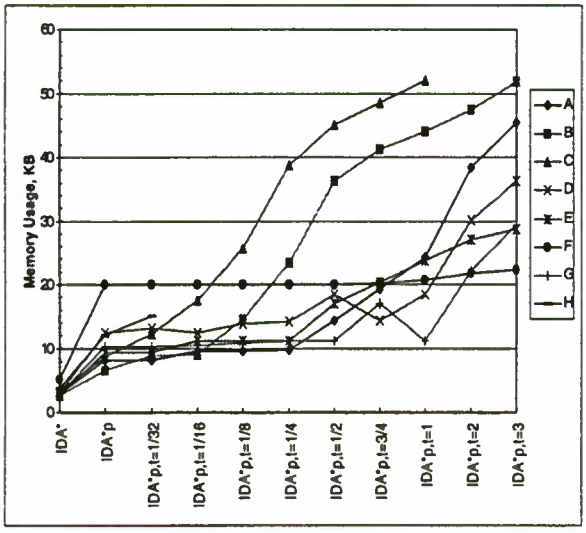


Figure 8: Memory usage (Kb) of IDA* with different thresholds

Memory usage is lowest for the plain IDA*, but increases across the other versions because reachability information to the neighbors is kept in the grid cells.

In contrast to run times, IDA*p, t=1/32 shows the least increase in memory usage on all but one scenario (D) whose t = 1/16 had slightly less memory usage. Figure 9 shows the memory usage of A* and IDA*p, t=1/32.

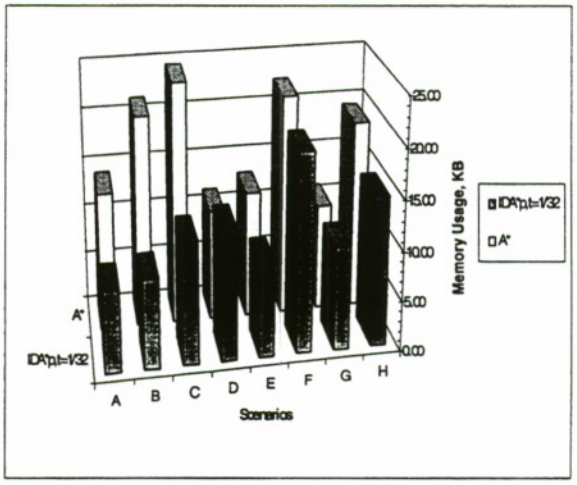


Figure 9: Memory usage (Kb) of A* and IDA*p, t=1/32

Table 7 summarizes the memory usage of A* and IDA*p, t=1/32.

Algorithm	Best	Worst	Average
A*	10.77	24.34	16.92
IDA*p, t=1/32	8.15	19.92	12.13

Table 7: Memory usage (Kb) of A* and IDA*p, t=1/32

On average, A* consumes 1.39 times as much memory as IDA*p, t=1/32.

6. Conclusions

This study compared the performance of A* and IDA* on graph search in a grid based terrain abstraction for route planning. The experiment showed A* to have faster run times but greater memory usage than IDA*'s.

Some techniques for optimizing IDA*'s run time performance were examined. These techniques yield run times that approach those of A*'s while increasing memory usage.

In Table 8, the algorithms are shown ranked in increasing order of run times.

Algorithm	Best	Worst	Average
A*	1.10	9.62	4.58
IDA*p, t=1/32	4.01	52.80	14.76
IDA*p	4.51	53.08	18.55
IDA*	22.09	424.78	161.55

Table 8: Summary of run times (seconds)

In Table 9, the algorithms are shown ranked in decreasing order of memory usage.

Algorithm	Best	Worst	Average
A*	10.77	24.34	16.92
IDA*p, t=1/32	8.15	19.92	12.13
IDA*p	6.58	19.96	10.94
IDA*	2.59	5.11	3.25

Table 9: Summary of memory usage (Kb)

Table 8 and 9 show the expected inverse relation between time and space efficiency.

Which algorithm is "better"? For graph searches, where memory usage is not a constraint (i.e., the search space is "small" or adequate memory is available), A* appears the better choice. When memory is a constraint, IDA* provides a memory efficient alternative. Because A*'s memory usage grows exponentially, IDA*'s linear growth rate is attractive. To obtain the best run time performance, two optimizations are available for IDA*. First, the neighbors (i.e., reachable locations) of each node should be stored in the search space rather than recomputed during search. Second, a small *advancement_increment* can be employed.

7. References

Karr C. R., and Rajput S. (1995). "Unit Route Planning", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida.

Loral (1994). "LibRoutemap documentation", Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1994.

Campbell, C. E. (1995). "Route Planning in CCTT", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida.

Korf, R. E. (1985). "Iterative-Deepening-A*: An Optimal Admissible Tree Search", *Proceedings of IJCAI-85*, Morgan Kaufmann, Los Altos, CA, 1985, 1034-1036.

Winston, P. H. (1992). *Artificial Intelligence*, Third Edition, Addison-Wesley Publishing Company, 1992.

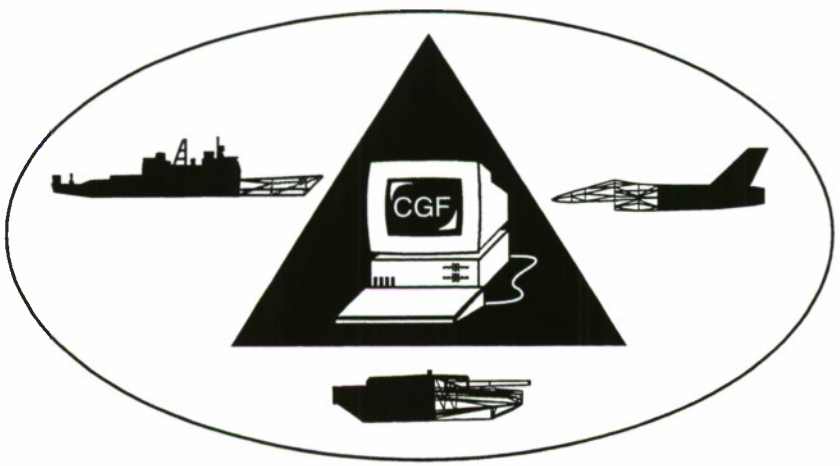
Even, S. (1979). *Graph Algorithms*, Computer Science Press, 1979.

8. Authors' biographies

Clark R. Karr is a Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Sumeet Rajput is an Associate Engineer in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science from the University of Central Florida. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Larry J. Breneman provides software support to the Institute for Simulation and Training. Mr. Breneman has a Master of Science degree in Computer Science from Bowling Green State University, Ohio.


CGF

Intervisibility Heuristics for Computer Generated Forces

Sumeet Rajput, Clark R. Karr, Mikel D. Petty, and Michael A. Craft
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
srajput@ist.ucf.edu

1. Abstract

Intervisibility between entities in a Distributed Interactive Simulation (DIS) environment is a mandatory, computationally expensive process. Computer Generated Forces (CGF) systems must frequently determine the intervisibility status between each of its controlled entities and each of the other entities in the simulation. Previous work has focused on developing algorithms to perform intervisibility determinations as quickly as possible. In this work, the problem was approached differently. Instead of speeding each intervisibility determination, heuristics were developed for reducing the number of determinations. The heuristics reduced intervisibility determinations with the savings ranging from 10% to 50%. The computational savings had negligible negative effect on the CGF entities' behavior with an average delay in sightings of 0.3 to 0.5. These results are independent of terrain representation and thereby applicable to any CGF system.

2. Intervisibility concepts

This section describes some of the basic concepts for intervisibility.

2.1 Definitions

Before the question "Does an entity see another entity?" is answered within a CGF system, the question "Is it possible for the entity to see the other entity?" must be answered? Typically, CGF systems' "sighting models" determine if entities "see" one another and incorporate the effects of attention, angle of view, weather, obscurants, time of day, and the details of sensor systems. However, sighting models are active only after "intervisibility" between two entities has been established. An intervisibility determination establishes or denies the existence of an unblocked Line of Sight (LOS) between two entities.

Establishing intervisibility between two entities involves checking to see whether terrain features (such as hills) or other objects (such as other vehicles) prevent a ray of light from traveling from one entity to the other. If a ray of light travels unhindered between two entities, the LOS is "unblocked". For the remainder of this discussion, the term "see" refers only to an unblocked LOS and does not imply that a sighting model has determined that an entity has sighted another. The notation A $\mapsto$ B will be used to mean A "can see" B. Note that intervisibility, as used in this discussion, is symmetric; if A can see B then B can also see A.

The phrase "intervisibility update" (IU) is used when an entity determines which of its opponents it can see. If there are n opponents in a scenario, A will do n individual (i.e., A-to-B) intervisibility determinations as part of an IU. The periodicity of IUs is determined by the "intervisibility update rate" (IUR).

Whenever the intervisibility status changes (from blocked to unblocked or vice-versa) an intervisibility "transition" is said to occur.

2.2 Computational methods and cost

Since extensive terrain checks are required to determine intervisibility status (blocked or unblocked) and the checks themselves are complicated, it comes as no surprise that the intervisibility computation taxes a system's resources. Very high intervisibility update rates can load a CGF system so much that the generated vehicle behavior degrades. Thus, it is important that the time spent in performing intervisibility determinations be reduced without sacrificing the tactical behavior of vehicles.

2.3 Statement of the problem

The goal of the research described in this report is to reduce the total computational load of intervisibility determinations on a CGF system. The reduction is to be achieved in a manner that has a minimum impact on the realism of the CGF entities' behaviors generated by the system.

IST's previous intervisibility research focused on efficiently performing each intervisibility determination within a polygonal terrain database (Petty et. al. 1992b). For the current work, an attempt

was made to reduce the number of intervisibility determinations made by a CGF system. The approach taken was to design, implement, and evaluate heuristics which would decide which intervisibility determinations could be skipped or delayed without affecting the CGF entities' behavior. See (Rajput et. al. 1994) for a complete description of this project. Because this heuristic approach is independent of terrain database format, the heuristics described and evaluated herein can be used in any CGF system.

3. Intervisibility in the IST CGF Testbed

Under the sponsorship of ARPA and STRICOM, IST has been conducting research in the area of CGF systems, seeking to increase the realism and autonomy of CGF behavior. A key product of that sponsorship is the IST CGF Testbed. The IST CGF Testbed is a CGF system that provides an environment for testing CGF behavioral control algorithms (Danisas et. al. 1990, Gonzalez et. al. 1990, Petty 1992a, Smith et. al. 1992a, and Smith et. al. 1992b). This section discusses the way intervisibility is done in the IST CGF Testbed and the data structures used to track entity intervisibility status, i.e., who has intervisibility to whom.

3.1 Intervisibility determination algorithm

Algorithm F (Petty et. al. 1992b), the Grid edge traversal method algorithm, is used in the IST CGF Testbed for computing intervisibility between entities. The CGF Testbed's internal terrain database format is a polygonal database. The surface of the terrain is represented in the database by contiguous non-overlapped polygons; the 3D vertices of each polygon are used to compute the height of any point within the polygon. The polygons are represented as edge lists and vertex lists within 125 meter^2 grids.

To determine intervisibility all edges of the polygons in the grids containing the LOS are checked for intersection with the LOS in two dimensions. If an edge/LOS intersection is found, the algorithm calculates whether the intersection lies above or below the LOS. If above, the LOS is blocked by the polygon edge; otherwise is unblocked by that edge. The number and complexity of calculations produce a time consuming computation.

3.2 Sightings list and intervisibility update duration

Each entity maintains a *sightings list*, which is a doubly linked list containing SIGHTINGS_ENTRY records. Each SIGHTINGS_ENTRY record describes the entity on whom sighting is being done, and intervisibility status (ERROR, INVISIBLE, DETECTED, RECOGNIZED and IDENTIFIED.).

During an IU, an entity does point-to-point intervisibility determinations to all target entities within visual range. Depending on the result of the point-to-point intervisibility determination (can the target entity be seen?) the status of the target entity is updated. New entities (those not already on the sightings list) are added to the list. Target entities which have remained invisible for a time greater than the "sighting persistence" limit are removed from the sightings list.

Receipt of an IU message triggers an entity to do its intervisibility update. When the intervisibility update is completed, the entity sends itself another IU message. Table 1 describes the terms used in connection with intervisibility updates.

Term	Description
IUD	Intervisibility Update Duration. This is the time between successive intervisibility updates and can vary as the simulation proceeds. In the IST CGF Testbed this is measured in one hundredths of a second. For example, an IUD of 25 means that intervisibility updates would be done every 0.25 seconds.
IUR	Intervisibility Update Rate. This is the frequency of the intervisibility updates and is the inverse of IUD. Thus, an IUD of 25 yields an IUR of 4 (i.e., 4 updates per second).
BUD	Base Update Duration. The IUD is initially set to this value which is read from a configuration file (sim.lod). The BUD is a constant for the simulation and is also measured in one hundredths of a second.
BUR	Base Update Rate. This is the initial frequency of the intervisibility updates and is the inverse of the BUD

Table 1: Intervisibility update durations and rates

Ideally, an IUD of 25 would cause an entity to do 4 intervisibility updates per second (an IUR of 4), while an IUD of 400 would, ideally, cause the entity to do 1 update every 4 seconds (an IUR of 0.25). If we are using an IUD of 100, we expect updates to be done at t+1 seconds, t+2 seconds, etc. The "ideal" IUR is always a maximum because of the systems clock granularity and various kinds of system overhead.

Although at lower IURs (i.e., high IUDs) delays due to system overhead are insignificant, one type of heuristic, Continuous Intervisibility Determination Avoidance (CIDA), Section 4.1.5, requires precise message delays because of the high message rates involved. In order to achieve the message rate for these heuristics, the software makes an adjustment to the message delay.

4. Intervisibility heuristics

This section discusses the types of heuristics that were implemented and their implementation details.

4.1 Types of heuristics

Generally, intervisibility heuristics are of two types: *physical* and *behavioral*. Physical heuristics attempt to reduce the number of intervisibility determinations by using some physical characteristic of intervisibility whereas behavioral heuristics exploit vehicle behavior.

A good example of a physical heuristic is the symmetry heuristic (Section 4.1.2). It is based on the physical nature of light that if A can see B then B can see A. Thus, the physical nature of light is the basis behind this heuristic.

Behavioral heuristics attempt to reduce the intervisibility determinations by using some behavior being done by an entity. The coarse-grain and fine-grain heuristics are all behavioral heuristics.

4.1.1 Varying the base update rate

The frequency of intervisibility updates is controlled by the intervisibility update duration (IUD). The IUD is initially set to the base update duration (BUD).

In battles fought in obstructed or hilly terrain, a low IUR may have a deleterious effect on the scenario. Sightings that could have been possible during periods of brief intervisibility may be missed. Hence, entities that could have been destroyed may survive to change the coarse of the battle.

4.1.2 Symmetry heuristic

As already noted, intervisibility is symmetric and, equally important, lack of intervisibility is also symmetric. An obvious heuristic is to inform B when A determines that $A \mapsto B$. B keeps this result in a scratch area (an area in memory containing historical and heuristic specific information). When B does an intervisibility determination, it first consults the scratch area for an $A \mapsto B$ result. If the interval between $A \mapsto B$ and "now" is "small", the result supplied by $A \mapsto B$ is accepted. Otherwise, it is assumed that A or B has moved enough to invalidate the determination and $B \mapsto A$ is determined.

4.1.3 History heuristic

It is often the case that when the outcome of an experiment or observation is consistent over time, people begin to assume the next outcome or observation will be the same.

For the problem at hand, it is reasonable to assume that if $A \mapsto B$ for a "long" time, then it is likely that $A \mapsto B$ the next intervisibility update. Similarly, if A doesn't see B for a "long" time, then it is likely the A will not see B the next intervisibility update.

The history heuristic is based on this idea. The heuristic tracks the number of consecutive intervisibility determinations that have returned the same intervisibility value. When a threshold value is passed, intervisibility determinations are skipped. When a transition is made, determinations are not skipped until a sufficient history accumulates.

4.1.4 Discrete intervisibility determination avoidance (DIDA) heuristics

The idea is to reduce intervisibility computation by skipping some of the intervisibility determinations for an entity. The role of the heuristic is to decide when to skip. DIDA heuristics are also known as coarse-grain heuristics.

For DIDA heuristics, the IUR, which had been initially set to the BUR, is not modified. Instead, intervisibility rate reduction is accomplished by skipping selected A-to-B intervisibility determinations.

Boolean heuristics are desired for discrete intervisibility determinations. The only choice is either to skip or not to skip the A-to-B intervisibility determination under consideration.

The main fault with this technique is its granularity. If the BUR is set as low (i.e., as infrequent) as is practical for realistic behavior, any skip (unless cunningly selected) is apt to cause behavior deterioration due to missed sightings. A natural tendency is to increase the BUR when discrete avoidance techniques are employed.

4.1.5 Continuous intervisibility determination avoidance (CIDA) heuristics

With CIDA heuristics, the interval between intervisibility determinations (IUD) is not necessarily an integer multiple of the BUD. The heuristics guess how long it is safe to wait until the next intervisibility determination based on the behavior of the entities in the simulation. CIDA heuristics are also known as fine-grain heuristics.

An optimal continuous avoidance algorithm would require separate IU messages for each A-to-B intervisibility determination being considered. This approach was not considered for this project because of the considerable programming and run time overhead involved.

Another approach is to adjust the interval between IU messages but this has the disadvantage of delaying a complete intervisibility update by an entity in order to delay a specific A-to-B intervisibility determination.

One practical technique is to approximate continuous delays for individual intervisibility determinations by using a relatively high IUR (in comparison to the BUR) in combination with the discrete techniques. This "fine-grain" approach is supported in these experiments. With suitable parameter adjustments (no algorithmic changes), the fine-grain technique will delay most intervisibility determinations most of the time and "intelligently" select which A-to-B intervisibility determinations should be applied for a given IU message. With a high IUR, a reasonable approximation of arbitrarily selected delays between A-to-B intervisibility determinations is feasible.

CIDA heuristics should yield a continuum of values. For convenience, all such functions are constrained to yield a range from zero to one. Zero indicates a minimum delay should be used (another intervisibility determination is needed *soon*), whereas one indicates the next intervisibility determination may be delayed by the maximum allowed interval (see Section 4.2).

The IUR should be high enough to approximate continuous delays substantially better than the BUR. But care should be taken that a high IUR does not overload the system with IU messages. In these experiments the IUR is obtained by multiplying the BUR by a *RATE* factor. This value indicates the number of times intervisibility checks are actually requested by the system as a multiplier of the BUR.

4.1.6 Composite heuristics

The composite heuristics are composed of sub-heuristics which vote whether to do an intervisibility determination. Each sub-heuristic computes a *metric* (*M*) value based on certain characteristics of the current simulation state. The computed metrics for the various sub-heuristics are used to determine a weighted average. If the weighted average exceeds a *threshold*, an intervisibility determination is skipped.

Sighter and target-based heuristics (Table 2) lend themselves to being composite heuristics. Composite heuristics can be either discrete (coarse-grain) or continuous (fine-grain). The three letter abbreviation for each heuristic appears in Table 2.

	Characteristic		
Heuristic	**Composite**	**Discrete (coarse-grain)**	**Continuous (fine-grain)**
Varying the BUR	No		
Symmetry	No	Sym	
History	No	His	Fgh
Sighter-based	Yes	Sgt	Fgs
Target-based	Yes	Trg	Fgt

Table 2: Implemented heuristics and their characteristics

4.2 Update rate limits for heuristics

Although DIDA and CIDA heuristics could make recommendations for arbitrarily long delays between updates, none of the heuristics designed could be expected to correctly predict long transition free periods. Thus, all the heuristics use a minimum sighting rate of *half the user's requested rate* (BUR). Given a BUR of 4.0, this ensures that after a sighting at t_0 another sighting will occur no later than (t_0+0.5) seconds.

4.2.1 DIDA heuristics

The coarse-grain heuristics receive IU messages at the user specified BUR. These heuristics ensure that only one intervisibility determination is skipped for any two consecutive messages. Thus, for a BUR of

1.0, if an intervisibility determination is done at t_0, the next determination will occur at either (t_0+1.0) or (t_0+2.0).

4.2.2 CIDA heuristics

This section discusses the minimum, maximum, and possible IURs for CIDA heuristics.

4.2.2.1 The minimum update rate for CIDA heuristics

The intervisibility scratch area keeps the last time an intervisibility determination was accomplished. When an IU message arrives, this area is checked and, if necessary to avoid too great a gap between updates, an intervisibility determination is forced. When this happens in a fine-grain heuristic, the heuristic is invoked to determine a new interval before the next determination.

4.2.2.2 The maximum update rate for CIDA heuristics

Although it is reasonable to hope heuristics could predict times when an accelerated rate would be beneficial, that is not a goal of this project. To this end, heuristic's recommendations are *always bounded above by the user requested rate* (BUR).

4.2.2.3 Possible update intervals for CIDA heuristics

For all the experiments completed, the fine-grain heuristics received IU messages at four times the BUR. So, for a BUR of 1.0 messages per second IU messages were received (approximately) every 0.25 seconds.

4.3 Heuristics and message overhead

The fine-grain technique increases the message handling overhead by increasing the number of IU messages sent. The finer the granularity, the greater the overhead.

Only the addition of local vehicles, vehicles created on this Simulator, adds to the message load. The number of messages per unit time is easily computed as the message rate per vehicle times the number of vehicles.

The fine-grain heuristics are designed to increase the message rate above the BUR according to a multiplier. When this parameter is 1 no additional messages are generated. When it is R, the requested rate will approximate R times the user requested rate. For these experiments, R was set to 4 (see Section 4.2.2.3).

4.4 Implemented heuristics

The heuristics implemented are:

- Varying the intervisibility BUR,
- The symmetry heuristic,
- The history heuristic (coarse and fine-grain), and
- The composite heuristics (coarse and fine-grain)

4.4.1 Varying the intervisibility base update rate

This is implemented as discussed in Section 5.1.1

4.4.2 Symmetry heuristic

The symmetry heuristic is implemented as discussed in Section 5.1.2.

4.4.3 History heuristic

For a discussion of the history heuristic see Section 5.1.3. This section discusses the mechanism that recommends whether to skip or not.

Both the coarse-grain and the fine-grain version of the history heuristic track the number of consecutive intervisibility determinations which have returned the same intervisibility result. In the coarse-grain version, the history of identical intervisibility values are compared to a threshold value. When the threshold value is exceeded, intervisibility determinations are skipped effectively reducing the update rate. When a sighting transition occurs, skips are inhibited until a sufficient history accumulates.

The fine-grain version calculates a skip value after a sufficient sighting history accumulates; this value refers to the number of intervisibility determinations that can be safely skipped between the sighter and a target.

The formula used to calculate the skip value in the fine-grain version is:

$$Skip = RATE - 1 + (MAX_SKIP - RATE + 1) \bullet \frac{Matches}{Interval}$$

where:

Skip	is the number of fine-grain intervisibility determinations to be skipped for this sighter/target pair
RATE	is the fine-grain multiplier (IUR = *RATE* · BUR)

MAX_SKIP is the maximum number of skips to preserve the minimum update rate (*MAX_SKIP* = 2 · *RATE* - 1)

Matches is the number of consecutive intervisibility determinations yielding the same result

Interval is the "full confidence interval". Once *Matches* equals *Interval* the minimum update rate is used.

When there have been no matches, Skip will be *RATE* - 1, which yields an IUR equal to the BUR. If the number of matches is as great as the interval, a heuristic parameter, the maximum number of skips will be done.

4.4.4 Composite, sighter-based, and target-based heuristics

Each component of a composite heuristic always computes a metric value (*M*) based on certain characteristics of the current simulation state. The metric computed by each sub-heuristic lies in the interval [0,1]. Composite heuristics occur in both the coarse-grain and the fine-grain versions.

4.4.4.1 Sighter-based heuristics

Sighter-based heuristics attempt to reduce the number of intervisibility determinations done by an entity by taking its behavior into account. The behavior may be some physical action of the entity, such as being stationary, or it may be some abstract behavior, such as having permission to fire.

Four sighter entity behaviors (sub-heuristics) were characterized for this study:

1. The movement of the sighter,
2. The sighter's permission to fire,
3. The sighter's ability to fire, and
4. The proximity of the sighter to enemy entities

The four sub-heuristics are assigned weights to increase or decrease their effects in deciding whether an intervisibility determination is required. A weighted average is computed to decide whether to do an intervisibility determination. In all cases, 0.0 indicates to do an intervisibility determination and 1.0 indicates NOT to do an intervisibility determination.

Any weight can be assigned to a sub-heuristic. The heuristic computes a metric for each behavior which is then multiplied by the weight assigned to it. For some behaviors this metric may be a boolean metric (0 or 1). For example, does an entity have permission to fire? For other behaviors this metric may be a floating point value; for example, the metric associated with the distance of a sighter to an enemy entity may be large (small) if the sighter is near (far) from the enemy entity.

The weighted average of the metrics for all the behaviors is guaranteed to be in the interval [0,1]. The weighted average of the metrics is given by the following equation.

$$Weighted_average = \frac{\sum_{i=1}^{4} wi * Mi}{\sum_{i=1}^{4} wi}$$

where:

wi is the weight assigned to a sub-heuristic *i*

Mi is the metric computed by sub-heuristic *i* (*Mi* is in the interval [0,1])

For the coarse-grain heuristics, judging whether or not to skip an intervisibility determination requires that a split point (threshold) be determined within this range. If the weighted average is greater than the split point, an intervisibility determination should be skipped, whereas a value less than the split value requires an intervisibility determination.

The fine-grain sighter heuristics are precise analogs to their coarse-grain versions. The key difference is that the composite vote of the heuristic elements no longer needs to be binary (no split point is needed) because the intervisibility determination is delayed by skipping intermediate fine-grain intervisibility determinations. In this case the number of fine-grain intervisibility determinations skipped is given by:

$$Skip = RATE - 1 + (MAX_SKIP - RATE + 1) \bullet Weighted_average$$

where:

Skip, RATE, and MAX_SKIP are the same as in Section 5.4.3 and

Weighted_average is the weighted average of the metrics of each sub-heuristic .

A weighted average of 0.0 will yield the maximum check rate (*RATE*-1) while a value of 1.0 will yield the minimum rate (*MAX_SKIP*).

4.4.4.1.1 Moving and stationary

This sub-heuristic is based on the premise that moving entities need to check intervisibility more

often than stationary entities. This sub-heuristic requests the minimum rate for a stationary vehicle and the maximum rate for a vehicle moving at its "normal speed".

$$M_1 = \frac{normal - current}{normal}$$

where:

normal is the normal speed of a vehicle
current is the current speed of the vehicle

When a vehicle is stationary, its "current speed" is 0 and the value of the metric, M_1, is equal to 1. This means that the sub-heuristic requires an intervisibility determination to be skipped. When the "current speed" equals the "normal speed", M_1 is 0 signifying that the sub-heuristic does not require any intervisibility determination to be skipped.

4.4.4.1.2 Permission to fire

Entities that have permission to fire should conduct more intervisibility determinations than entities that do not. This sub-heuristic falls into the category of boolean sub-heuristics.

$M_2 = 1.0$ if entity has permission to fire, 0.0 otherwise

4.4.4.1.3 Able to fire

The ability to fire may be lost by an entity after it has been hit by enemy fire. This sub-heuristic also falls in the category of boolean behaviors.

$M_3 = 1.0$ if able to fire, 0.0 otherwise

4.4.4.1.4 Proximity to target

It seems natural that an entity would do more intervisibility determinations when it is in the vicinity of enemy entities than when it is far away from them. An entity is considered "in the vicinity" of an enemy entity when it is lies within the maximum weapon range of the target entity.

As the entity or sighter gets closer to the enemy or target entity more intervisibility determinations are done.

$$M_4 = \frac{d}{r}$$

where:

d is the distance to the target entity
r is the sighter's maximum weapon range.

4.4.4.2 Target-based heuristics

Target-based heuristics reduce the number of intervisibility determinations done by a sighter to *a particular* target by taking into account the type, appearance and behavior of that target.

The implementation of the target based heuristics is very similar to the sighter-based heuristics except characteristics of the target are examined. Four target entity behaviors were characterized for this study:

1. The relative movement of the sighter and target
2. The estimated threat of the target
3. Target damage status and
4. The proximity of the sighter to a target

4.4.4.2.1 Relative movement of sighter and target

The movement dimension to this heuristic uses the sighter's and target's velocities to determine the speed at which they are closing/separating. The rational is sighters need to more carefully watch approaching targets.

$$M_5 = 0.5 + \frac{R}{2 \bullet C}$$

where:

R is the rate the sighter and target are closing (in m/sec.), expected range $[-C, C]$
C is the closing rate for the maximum update rate (12 m/sec. in these experiments)

If vehicles are separating rapidly $R \leq -C$, $M_5 = 0$. If they are closing rapidly, $R \geq C$, $M_5 = 1$.

4.4.4.2.2 Estimated threat of target

Sighters do more intervisibility determinations to targets that are considered more threatening than to targets that are less threatening. A target's threat is determined by the priority the target has been given as a target for the weapons the sighter carries.

M_6 = 1.0 if target is NOT first, second or third priority; 0.0 otherwise

4.4.4.2.3 Target damage status

Damaged targets are given less attention than healthy targets.

M_7 = 1.0 if target has firepower kill or is destroyed; 0.5 if it has mobility kill; 0.0 otherwise

5.4.4.2.4 Proximity to target

Entities check more often when an enemy entity is relatively close. Close is computed in terms of the target's firing range, and the amount of attention paid is proportional to its distance.

$$M_8 = \frac{d}{r}$$

where:

d is the distance to the target entity

r is the maximum range of any weapon possessed by the target entity

This behavior is similar to the case in the sighter-based heuristics where a sighter varied its IUR to a target depending on their relative distance. However, in one case the sighter's maximum weapons' range is used to determine its IUR while in the other, the target's maximum weapons' range is used.

In Figure 1 the circles represent the maximum range of weapons of entities A and B. Consider the sighter-based sub-heuristic with A being the sighter entity and B the target entity. A's IUR should not increase because B is beyond its (the sighter's) maximum range of weapons. However, if B is the sighter entity, B's IUR should increase.

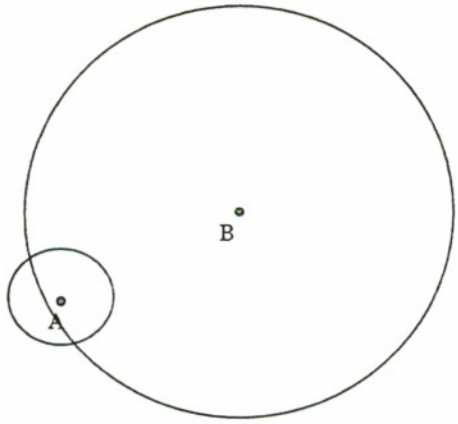


Figure 1: Sighter, target maximum weapons ranges

In contrast, consider the target-based sub-heuristic. When B is the target entity, A's IUR should increase. When A is the target entity, B's IUR should not increase.

5. Evaluation of the intervisibility heuristics

Evaluation of the heuristics requires:

- Establishing the performance metrics,
- Data collection, and
- Heuristic ranking based on effectiveness.

This section addresses these points in detail.

5.1 Evaluation experiment

The performance of the intervisibility heuristics was evaluated in the context of a set of three standard military scenarios (Rajput et. al. 1994).

The performance of the CGF Testbed Simulator in the area of intervisibility was measured and compared for heuristic and "base" versions in each scenario. The "base" versions were "no-heuristic" versions. Intervisibility performance was based on number of intervisibility determinations, sighting event times, and computational overhead.

A "sighting event" or "sighting" refers to the first detection of an unblocked line of sight between two entities for which there was no such line of sight just prior to the event. The "sighting event time" is the simulation time of such an event.

5.1.1 Performance metrics

The following data was gathered:

- The total number of sighting events,
- The sighting event time, the sighter and target IDs, and their locations, and
- The time, in clock ticks, at various stages of the intervisibility updating process.

For a scenario, the set of sighting events and sighting event times found by the no-heuristic version is taken to be the "true" or "correct" set. When a heuristic is compared to the no-heuristic version the following cases arise:

- Sighting events may be *missed* by a heuristic
- There may be *extra* sighting events in the heuristic output
- Sighting events may be *delayed*
- Sighting events may occur *earlier*

Some sightings will be missed or be extra because of sampling error. Using coarse BUDs (1/2 second or more) makes it inevitable that some transitions will be missed, both by the heuristic version (labeled "missed") and by the no-heuristic version (labeled "extra"). The real question is how many sightings are missed because of delayed checking. Extra sightings are always from sampling error since the heuristics never do more frequent checks than the no-heuristic version of the system. It seems likely (and experiments support this) that the greater the average sighting delay for the System Under Test (SUT) the more misses will be recorded.

It is possible that the computational cost of computing some heuristic may exceed the cost of doing a real intervisibility determination. Both the sighter and target-based heuristics are quite complex, particularly when all their components are active. It is, therefore, not enough to simply count the number of intervisibility determinations done. Instead, a comprehensive evaluation is needed that takes the computational cost of the heuristics in account.

Because the fine-grain heuristics produce a large number of internal messages, heuristics of this type must consider the message delivery overhead. Experiments revealed that message delivery time was a minor issue. Measurements yield delivery time on the order of 1/4 to 1/2 milliseconds (4000-2000 deliveries/sec.). On the other hand, overhead for heuristics is proportional to the number of targets. If l is the number of local vehicles and m is the number of remote vehicles, the heuristic overhead is proportional to their product $l*m$, but message delivery is proportional to l only.

With this in mind, the sighting delays and the computational overhead of a heuristic are used in measuring the "cost" of the heuristic.

5.1.1.1 Savings calculation

Naively, it may seem that the effectiveness of using a heuristic is the difference between the number of point-to-point intervisibility determinations expected by a user for the scenario with a particular setting of the BUD and the number of point-to-point intervisibility determinations actually done by a scenario. This is not true because the savings obtained may be optimistic as the heuristic overhead has not been taken into account. The overhead of the heuristic offsets the savings from a reduction in the number of intervisibility determinations. The net savings must be used to evaluate the quality of a heuristic.

The effectiveness of a heuristic is represented as the ratio of its savings to the cost the system has to incur to use it. Thus, we have:

$$E_{h,s} = \frac{S_{h,s}}{C_{h,s}}$$

where:

- $E_{h,s}$ is the effectiveness of heuristic h for scenario s
- $S_{h,s}$ is the savings achieved by heuristic h for scenario s
- $C_{h,s}$ is the cost of using heuristic h for scenario s.

Table 3 shows the Overhead Multiplier (OM) for each heuristic. The OM for a heuristic is a measure of the overhead associated with using that heuristic. It is defined as the ratio of the total time spent processing point-to-point intervisibility determinations between vehicles with the heuristic to the total time spent in doing these determinations without a heuristic.

Heur	OM
Sym	1.06
His	1.01
Sgt	1.03
Trg	1.05
Fgh	1.02
Fgs	1.005
Fgt	1.03

Table 3: Computing the overhead multiplier

where:

- Heur Name of the heuristic (refer to Section 5)
- Raw Number of point-to-point intervisibility determinations done
- Tics Total time spent by the SUT exercising intervisibility code
- t_{raw} Time that would be taken if no heuristic was being used to do the checks (t_{raw} = Raw / 53.9)
- OM Overhead multiplier (OM = Tics / t_{raw})

After the overhead multiplier has been determined, the savings $S_{h,s}$ can be determined.

$$S_{h,s} = (1 - (\eta * O_h)) * 100.0$$

where η is the number of sightings that heuristic h does in scenario s and O_h is the OM for heuristic h.

Example:

Assume a heuristic does 75% of the intervisibility determinations done by the no-heuristic version in the same scenario (an apparent saving of 25%). However, if it has an overhead multiplier of 1.05 the real saving is:

$$S_{h,s} = 1 - (0.75 \times 1.05) = 0.2125 \text{ or } 21.25\%$$

5.1.1.2 Missed and extra sightings calculation

It may seem that a heuristic should be penalized for "missed" sighting events. A sighting event is

considered "missed" by a heuristic if it failed to produce a sighting event found by the no-heuristic version. It may be argued that a heuristic must be "bad" if it misses many sighting events (and "good" if it does not). However, the situation is more complicated than that.

When a scenario is repeated results from the second run are not generally the same as in the first run. Small system perturbations from various non-deterministic events, such as network packet delivery times, have cumulative effects resulting in different intervisibility determination sampling (although the frequency is the same). It was found that a no-heuristic simulator run against the test scenarios showed approximately 5% variability from run to run in terms of missed and extra sightings.

It would be difficult to directly reflect the missed events in the computation of the heuristic's cost. One real difficulty to overcome was how to allow for the misses generated by heuristics which intentionally "missed" many sightings. Misses, other than those caused through sampling variation, are closely tied to the mean and standard deviation for the sighting delays. Heuristics that delayed sightings greatly are prone to miss sightings.

Analysis of the results of the combat scenarios (see Section 5.1.2), excluding the sighter and target data, revealed a positive correlation between the raw metric used for evaluation and the number of misses seen for the heuristic/scenario trial. The correlation was not very high (correlation coefficient was 0.374). The low correlation, the variability from run to run, and the difficulty of accounting for the misses led us to ignore this factor in evaluation.

5.1.1.3 Sighting delay calculation

The "sighting delay" is the difference in the simulation times of the same sighting event in the heuristic and no-heuristic version. The sighting delays must be a factor in determining a heuristic's cost (and hence its effectiveness). A heuristic should be penalized for delays; a heuristic that "sees" events earlier is preferable to another that "sees" them later. The absolute mean of the delays is used as one of the measures of the cost of using a heuristic.

One may argue that the mean of the signed delays should have been used instead of the absolute value. However, the heuristics are not designed to sight earlier. There may be some negative delays in sightings (i.e., sightings were done earlier) but these are generally due to sampling variations. A heuristic should not be given credit for sampling variations and so a signed value should not be used.

Another parameter used in the cost equation is the standard deviation of the absolute delays. If two heuristics have the same sightings delays, the heuristic having a smaller standard deviation is preferred over the other.

Although the cost of a heuristic should rise with the standard deviation, the standard deviation is deemed less important than the mean. To reduce the impact of the standard deviation its square root is used.

5.1.1.4 Heuristic cost calculation

Combining the cost parameters we obtain the following equation for the *raw measure* of the cost of using a heuristic *h* for scenario *s*.

$$R_{h,s} = \mu * \sqrt{\sigma}$$

where:

- $R_{h,s}$ is the raw measure of the cost of using heuristic *h* for scenario *s*
- μ is the absolute mean of the sighting delays induced by heuristic *h* for scenario *s*
- σ is the absolute standard deviation of the sighting delays

Table 4 displays the data used to compute the run to run variance, an important factor in calculating a heuristic's cost. The $|\mu|$ and the $|\sigma|$ values are obtained by running the no-heuristic version twice and comparing the data.

Scenario	$\|\mu\|$	$\|\sigma\|$	$R_{0,s} = \|\mu\| * \sqrt{\|\sigma\|}$
Meeting	0.21	0.34	0.122
Delay	0.31	0.32	0.175
Assault	0.26	0.39	0.162
Meeting (C)	0.30	0.26	0.153
Delay (C)	0.31	0.34	0.181
Assault (C)	0.29	0.31	0.161

Table 4: Computing run to run variance

The $R_{0,s}$ column indicates the run to run variance for each scenario. To compute the cost, $C_{h,s}$, for each heuristic/scenario pair, the raw measure of the cost $R_{h,s}$ of that heuristic/scenario pair is divided by the $R_{0,s}$ value for the scenario *s*.

$$C_{h,s} = \frac{R_{h,s}}{R_{0,s}}$$

where:

$C_{h,s}$ is the cost of using heuristic h for scenario s

$R_{h,s}$ is the raw measure of the cost of using heuristic h for scenario s and

$R0_{,s}$ is the nominal cost (variation) from run to run

After the cost of using a heuristic $C_{h,s}$ is determined, the effectiveness $E_{h,s}$ is computed by dividing the savings $S_{h,s}$ by the heuristic cost.

$$E_{h,s} = \frac{S_{h,s}}{C_{h,s}}$$

5.1.2 Experimental design

Three types of engagements were used to test the heuristics:

- Meeting
- Delay
- Assault

Six scenarios were developed; a *movement-only* and a *combat version* for each type of engagement. The experiment consisted of running an unmodified and heuristically modified versions of the Simulator with these scenarios and collecting data for analysis.

5.1.3 Data collection

A project of this complexity requires the analysis of large amounts of data. The scenarios that were developed for the evaluation of heuristics ran from 6 to 10 minutes. For each scenario, data was collected for 7 heuristics. Two runs were made without heuristics; one became the reference data, and the other a "base version" used to evaluate run to run variability.

Initially, data was collected by creating a point-to-point network (to reduce network processing) between two Simulators; each running its part of the scenario. This approach did not prove viable. Because of the ill conditioned nature of the experiment (Rajput et. al. 1994), a second run of a scenario would usually diverge from the first. By the end of a run of more than a few minutes the sighting event histories would be very different.

The problem of scenario divergence was solved by logging each scenario's network traffic. For the evaluation of a system, the logged data was replayed; the scenarios were recreated exactly in terms of the network activity. The logging process was automated to remove errors in synchronizing the start and end of the scenarios. Two personal computers ran the Simulators, while a third logged the network traffic. The experiment was isolated on a LAN.

To conduct an experiment, a scenario log was played back to generate test data for heuristic evaluation using a two PC point-to-point arrangement. One PC played the logged scenarios repeatedly, while the second PC ran a modified Simulator that did intervisibility tests between entities that were on remote machines.

5.2 Experimental results

All the components of the sighter and target-based heuristics, both the fine and coarse-grain versions, were given equal weights so all of the components could be exercised (refer to Section 4.4.4.1 and Section 4.4.4.2).

5.2.1 Overall heuristic performance

A heuristic's overall performance can be seen by comparing the range of its $E_{h,s}$ values. A good heuristic has high $E_{h,s}$ values and low variability. Heuristic A is said to be more stable than B if its effectiveness is independent of the scenario.

Figure 2 shows the range of $E_{h,s}$ values for the implemented heuristics, and $E_{h,s}$ values for intervisibility BURs of 0.67 (a check every 1.5 seconds) and 0.5 (a check every 2 seconds). This figure shows the history heuristic to be the most stable heuristic. Sighter and target-based heuristics have very large spreads, with the extreme left points showing negative efficiency. The fine-grain sighter and target-based heuristics are marginally more stable and effective then their coarse-grain versions. The symmetry heuristic showed the greatest effectiveness with stability second to the history heuristic.

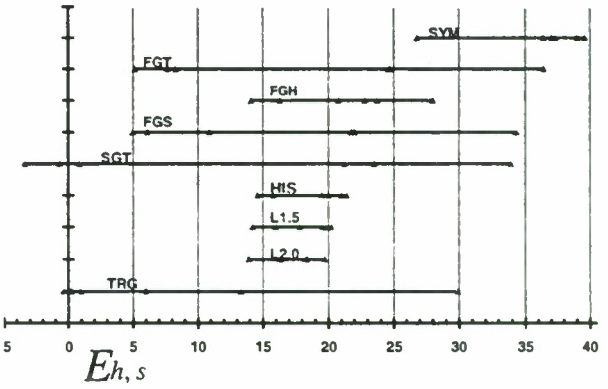


Figure 2: Heuristic performance spread

Interestingly, all heuristics performed well in vigorous situations. The combat scenarios appeared predominantly on the right of each line and the non-combat on the left.

To determine the overall effectiveness *Eh* of a heuristic a weighted average is used. Since heuristics have more to offer in the combat scenarios, more weight (3 times) is given to such scenarios than to non-combat scenarios. *Eh* is computed as:

$$E_h = \frac{(E_{h,s1}+E_{h,s2}+E_{h,s3})+3(E_{h,s4}+E_{h,s5}+E_{h,s6})}{(E_{h,s1}+E_{h,s2}+E_{h,s3}+E_{h,s4}+E_{h,s5}+E_{h,s6})}$$

where:

- *Eh* is the overall effectiveness heuristic h
- *Eh,x* is the effectiveness of heuristic h for scenario x
- *s1* is the Meeting scenario
- *s2* is the Delay scenario
- *s3* is the Assault scenario
- *s4* is the Meeting (Combat) scenario
- *s5* is the Delay (Combat) scenario
- *s6* is the Assault (Combat) scenario

Using this metric the heuristics are "ranked" in Table 5. The table also shows the overall savings, *Sh*, and the overall cost, *Ch*.

Heuristic	*Eh*	*Sh*	*Ch*
Symmetry (Sym)	37.0	46.4	1.3
Fine-grain target (Fgt)	23.2	29.7	1.30
Fine-grain history (Fgh)	22.0	35.1	1.7
Fine-grain sighter (Fgs)	21.3	25.5	1.2
Sighter (Sgt)	19.4	16.7	0.9
History (His)	18.9	45.8	2.5
BUR 0.67	18.3	33.3	1.8
BUR 0.5	17.6	50.0	2.8
Target (Trg)	12.4	16.2	1.2
BUR 0.33	10.9	66.6	6.2
BUR 0.25	7.2	75.0	10.5
BUR 0.2	5.1	80.0	15.8
BUR 2.0	-85.0	-100.0	1.2

Table 5: Heuristic rankings

Data was gathered by running the Simulator with different BUR settings (2.0, 0.67, 0.5, 0.33, 0.25, and 0.2) to see the effects of the IUR on the sightings. Table 5 shows that except for BUR 0.67 and BUR 0.5, the BUR-based heuristics performed poorly and are at the bottom of the rankings.

BUR 0.67 and BUR 0.5 seem to perform better than the target-based heuristic; but this is partly an illusion. In contrast to fixed BURs, the target-based heuristic (as well as all other heuristics) save "intelligently." Intervisibility determinations are delayed only when they are deemed acceptable, for example, when a scenario is calm. However, no delays are allowed when the scenario becomes more active, for example, when combat starts.

5.3 Evaluation comments

Even the least effective heuristic studied (coarse-grain target) saves almost 40% of the intervisibility determinations for some scenarios; for example, Delay with Combat. It does this with high effectiveness. It can be argued that combat situations are where the heuristics are most needed because this is the most typical use of CGF systems.

The sighter and target-based heuristics seem reasonable; for example, destroyed vehicles should not be sighted or attempt to sight. The other components of these composite heuristics are similarly reasonable, but time did not permit experimental validation of each component individually. The components taken together performed well.

Symmetry is shown to save on the order of 50% for all scenarios. This is a very intuitive result; because of the way symmetry works, one would expect it to eliminate half of the intervisibility determinations. The fact that this result was found in the experiment adds credibility to the experimental method used. Of course, the symmetry heuristic was tested only for entities that were generated by the same Simulator. Applying the symmetry heuristic across multiple Simulators (i.e., multiple network nodes) would require network traffic to communicate the symmetry results. It is not clear whether the reduction in intervisibility processing produced by the symmetry heuristic is worth the additional network processing.

6. Future work

Perhaps the biggest unanswered question, and, thereby, an opportunity for future work, is what might happen if the different heuristics were combined? Time constraints did not permit the examination of this issue in the project. Symmetry especially seems

to be a likely candidate for combination with other heuristics , as its basic idea is very different from the other heuristics.

7. Conclusions

Computer generated forces are becoming increasingly complex as additional functionality is being added and more realistic behaviors are expected. It makes sense that some of the computational load be removed so that the CGF system can give more time to processing additional functions.

For this project, a number of intervisibility heuristics were designed, implemented, and experimentally evaluated within a CGF system. The overall goal of the heuristics was to reduce the overall computational expense of intervisibility determination in CGF systems without materially affecting the realism of the autonomous behavior produced by those systems. The results show that the implemented intervisibility heuristics save substantial portions of the processing devoted to intervisibility checks, ranging from 10% to 50%.

Moreover, these savings were achieved at little cost in terms of CGF behavior realism. The behavior generated by the CGF system would be expected to suffer if an intervisibility heuristic significantly delayed the times at which hostile entities were sighted. That did not occur; the average sighting delay imposed by the various heuristics fell in the range of 0.3 to 0.5 seconds. Such a delay is negligible, especially in light of the tremendous savings in processing.

These results can be of great importance to CGF systems. By using one of these heuristics, the computational load of intervisibility determination can be greatly reduced, thereby freeing computational capacity that can be applied to generating more sophisticated behavior, performing more realistic physical modeling, or simply controlling more entities on a given system. Because these heuristics are independent of the terrain database format, they can be applied to any CGF system. Therefore, one or more intervisibility heuristics should be seriously considered for inclusion in any real-time CGF system.

8. Acknowledgment

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

9. References

DIS Steering Committee (1993). "The DIS Vision: A Map to the Future of Distributed Simulation", *IST Technical Report*, 47 pages.

Loper, M. L., Thompson, J. R., and Williams, H. L. (1991). "Simulator Networking: What Can It Offer The Training Community?", *Military Simulation & Training*, Issue 4 1991, pp. 11-14.

Danisas, K., Smith, S. H., and Wood, D. D. (1990). "Sequencer/Executive for Modular Simulator Design", *Technical Report IST-TR-90-1*, Institute for Simulation and Training, University of Central Florida, 16 pages.

Gonzalez, G., Mullally, D. E., Smith, S. H., Vanzant-Hodge, A. F., Watkins, J. E., and Wood, D. D. (1990). "A Testbed for Automated Entity Generation in Distributed Interactive Simulation", *Technical Report IST-TR-90-15*, Institute for Simulation and Training, University of Central Florida, 37 pages.

Petty, M. D. (1992a). "Computer Generated Forces in Battlefield Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 56-71.

Smith, S. H., Karr, C. R., Petty, M. D., Franceschini R. W., and Watkins, J. E. (1992a). "The IST Computer Generated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, University of Central Florida.

Smith, S. H., and Petty, M. D. (1992b). "Controlling Autonomous Behavior in Real-Time Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 27-40.

Petty, M. D., Campbell, C. E., Franceschini, R. W., Provost, M. H., and Karr, C. R. (1992b). "Preliminary Investigations into Efficient Line of Sight Determination in Polygonal Terrain", *Technical Report IST-TR-92-5*, Institute for Simulation and Training, February 28 1992.

Rajput S., Craft, M. A., Breneman, L. J., Petty, M. D., Karr, C. R., Holly, T. P., Ng, J. J. (1994). "Intervisibility Heuristics for Computer Generated Forces", *Technical Report IST-TR-94-22*, Institute for Simulation and Training, University of Central Florida.

10. Authors' Biographies

Sumeet Rajput is an Associate Engineer in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Clark R. Karr is the Computer Generated Forces Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He currently manages the Plowshares emergency management simulation project; previously he led Computer Generated Forces research at IST. Mr. Petty has a B.S. in Computer Science from the California State University Sacramento and a M.S. in Computer Science from the University of Central Florida. His research interests are in simulation and artificial intelligence.

Michael A. Craft is a Research Associate working for the Intelligent Simulated Forces project at the Institute for Simulation and Training. He has a Master of Science degree in Mathematics and a Master of Science in Computer Science. His research interests are in the areas of Software Engineering, Real Time Systems, Simulation, and Network Protocols. Mr. Craft has a broad and diverse background ranging from Office Automation to Missile Tracking Systems.

Benchmarking and Optimization of the IST CGF Testbed

Stephen A. Schricker, Tracy R. Tolley, Robert W. Franceschini
Institute for Simulation and Training
3280 Progress Drive, Orlando, Florida 32826
sschrick@ist.ucf.edu

1. Abstract

This paper discusses the establishment of a performance benchmark and the current capacity of the IST CGF Testbed (henceforth called the Testbed). It also describes the manner in which the Testbed manages and subsequently measures its simulation load, and relates a particular means of load measurement to that which has been standardized for ModSAF. It then introduces additional methods for benchmarking those features that are unique to the Integrated Eagle/BDS-D project, which uses the Testbed as a component. In addition, this paper presents some benchmarking results of scenarios for which a corresponding benchmark of ModSAF is not currently available.

2. Background

The Integrated Eagle/BDS-D project (Franceschini 1995) (Karr 1994) links a constructive, aggregate-level simulation (Eagle) with a virtual, entity-level simulation (DIS/SIMNET). The linkage uses the Testbed to control those vehicle platforms that are instantiated in the virtual simulation as the components of aggregate units in the constructive simulation.

Network bandwidth and computer processing power severely limit the number of vehicle platforms (or entities) that can participate in existing DIS/SIMNET exercises. Therefore, one of the major goals of a constructive+virtual linkage (Franceschini 1995) is to manage this limitation. The constructive simulation provides the context of a large-scale battle (say, at the Corps or Division level) while the virtual simulation plays out the smaller engagements. Such smaller engagements, however, need to be large enough for the battle to have any intrinsic meaning. Consider, for example, a scenario involving helicopters on a reconnaissance mission. In order for the small engagement occurring in the virtual simulation to include the helicopters, the virtual environment must also contain many other disaggregated units from the constructive environment, as a helicopter's sensor systems typically allow it to detect and engage vehicles from over four kilometers away. To ensure the realism and usefulness of the constructive+virtual linkage, the CGF component of the virtual simulation must therefore have a large entity capacity.

IST developed its CGF Testbed as an inexpensive, experimental simulation engine that could run on IBM-compatible personal computers operating under DOS. As such, and written entirely in C, it was developed under the Borland C environment, version 3.1. The Testbed developers ultimately discovered, however, that as a DOS application, the Testbed was constrained to the first 640 kilobytes of internal RAM on a personal computer that DOS considers conventional memory. This effectively limited to twelve the maximum number of entities that could be simulated on a single personal computer. The ever-increasing power of personal computer processors, however, suggested that the Testbed might support many more entities.

In an effort to increase the capacity of the Testbed, IST's Integrated Eagle/BDS-D project team converted the Testbed to the WATCOM development environment, version 10.0, which in conjunction with Rational Systems' DOS/4GW DOS extender, gives DOS applications access to as much as four gigabytes of RAM in a flat address space (WATCOM 1994).

2.1 Why Benchmark the Testbed Now?

It has now become evident that by giving the Testbed potential access to every bit of free memory on the computer, the limiting factor in the performance of the Testbed may be the processor itself. Thus it seems an opportune time to benchmark and standardize the performance of the Testbed to determine whether this is actually the case. As an added benefit, the methods for benchmarking the Testbed will not only show just how large an entity load it can handle, but also provide an objective means of gauging the system's performance under any conditions.

In an effort to produce valid, standard results, the methods used to benchmark the Testbed were similar to those described in (Vrablik 1994) in the benchmarking of ModSAF.

3. How Do ModSAF and the Testbed Differ?

Both ModSAF and the Testbed perform basically the same function, but minor differences in the methods they use become more apparent under closer analysis.

3.1 Simulation Hosts and Operator Interfaces

One important difference between ModSAF and the Testbed is that ModSAF consists of a single executable application, though it can take on different forms through the use of the available command-line options. In this manner, ModSAF can run as either a ModSAF Station (SAFStation) or a ModSAF Simulator (SAFSim). A SAFStation can run as an operator interface, a simulation host, or a "pocket" system: an integrated operator interface/simulation host. A SAFSim can run only as a simulation host. It is possible to network several SAFSims and SAFStations in order to increase the entity capacity of an exercise (ModSAF 1994).

The Testbed, on the other hand, is composed of separate executable applications: the simulation host, which performs the virtual-level simulation of entities; and the operator interface, which provides a graphical interface for the user. Though the simulation host does have a rudimentary operator interface, the Integrated Eagle/BDS-D system utilizes the dedicated operator interface exclusively as its user interface.

In any case, there is a distinct advantage to running the simulation host and operator interface separately. Though it is possible to develop a single integrated application which is able to perform all of these functions (as the SAFStation can), under a heavy load, such an application spends most of the time processing its own overhead (Vrablik 1994). Running the simulation host and the operator interface separately maximizes the amount of processor resources devoted to performing each task. Of course, this type of system requires more hardware, but since the Integrated Eagle/BDS-D system uses IBM-compatible personal computers, the cost of gathering the requisite hardware (for Eagle/BDS-D, at least) is relatively small.

3.2 How Does ModSAF Simulate its Entities?

As with most virtual-level simulation hosts, ModSAF and the Testbed are both sequential in nature. In other words, they can process only one thing at a time. This presents a special challenge when developing an application that simulates the behaviors of numerous entities. The developer must ensure that the execution of any one task does not overtly compromise the integrity of the simulation.

To this extent, ModSAF utilizes a task scheduler, which keeps a list of the jobs it needs to do, including running each vehicle it is simulating (Vrablik 1994). The scheduler uses, as its basic data structure, a heap-based priority queue. The ModSAF scheduler is non-preemptive–it allows each task to finish before moving on to the next. For more information on ModSAF's task scheduler, refer to the LibSched documentation in the *ModSAF Programmer's Reference Manual* (ModSAF 1994)

The tasks that ModSAF must perform in updating a vehicle are based on what that vehicle is doing at that particular time. It may be a single, simple task, such as broadcasting the vehicle's position over the network; or there may be multiple, complex tasks, such as performing complicated movements, performing intervisibility and detection calculations on other potential targets, tracking a target, and firing a weapon (Vrablik 1994). Thus, the amount of processor time devoted to a single update of a single vehicle may be either quite small or relatively large, depending on what that vehicle is doing.

3.3 How Does the Testbed Work?

Likewise, the Testbed's simulation host uses a message scheduler to oversee its tasks. Like ModSAF's task scheduler, the Testbed's message scheduler is implemented as a priority queue. Also, like ModSAF, the Testbed's message scheduler is non-preemptive. The most significant difference between the implementations of ModSAF's task scheduler and the Testbed's message scheduler, however, lies in what happens during the processing of a single task.

Essentially, a task in ModSAF's task queue and its analog, a message in the Testbed's message queue, is a request for access to the processor to perform the specified task. When a vehicle in ModSAF requests access to the processor to perform an update, it updates all of its systems at once–from its position, heading, and orientation, to tracking and firing at a target, to whatever else it may need to update at the time. Thus, there may be a large difference in the amount of processor time devoted to the updating of two different vehicles, depending on what those vehicles are doing.

Messages in the Testbed, however, are much more specific. For example, a vehicle in the Testbed may request access to the processor to update its position, heading, and orientation. To update its target

acquisition list or to fire a weapon, for example, the vehicle would place a different request onto the message queue corresponding to the respective activity. In essence, whereas a single task in ModSAF might be "update vehicle number 100," the analog in the Testbed would be "update the position of vehicle number 100," or "update the damage incurred on vehicle number 100," and so forth. Thus, even though there is still a difference in the amount of processor time devoted to the performance of different tasks, the amount of processor time devoted to the performance of any single task is much smaller.

Each of these methods of updating vehicles has its own advantages and disadvantages, though this is not the subject of this paper.

4. Metrics for Measuring CGF Performance

The process in ModSAF of going through the scheduler's list of vehicles and tasks once is called a "tick," and the process of handling a particular vehicle's requirements during a single pass through the local vehicle list is called "ticking" that vehicle (Vrablik 1994). One interpretation of this definition might be that a tick consists of a single access to the processor for any particular vehicle. This seems a most likely application of the definition when attempting to benchmark the Testbed due to the system's underlying architecture. This application, however, will yield very different results when gauging the system's performance.

4.1 Not All Ticks Are Created Equal

When ModSAF ticks a vehicle, everything about that vehicle is updated at once. Therefore, depending upon what a vehicle is doing, there may be quite a large difference in the amount of processor time devoted to updating two distinct vehicles. However, during the course of a simulation, ModSAF will tick each vehicle the same number of times.

A single update of any particular vehicle in the Testbed consists strictly of updating only one system on that vehicle, such as the vehicle's movement, target acquisition, tracking of targets, or weapons firing. If we determine that each of these actions comprises a single tick, it becomes apparent that there will be a smaller difference, compared to ModSAF, in the amount of processor time devoted to a single update for two different vehicles. This implies that the Testbed will tick more frequently those vehicles that are more active.

4.2 The Underlying Problem

(Vrablik 1994) describes a method of benchmarking ModSAF by measuring the average interval between ticks for individual vehicles. As the number of simulated vehicles increases, the load on the system also increases. Hence, the frequency with which the system ticks a particular vehicle decreases. There exists a point where the time between individual vehicle ticks becomes large enough that it begins to affect the behaviors and movements of the simulated vehicles. Historically, this threshold tends to occur at tick lengths of about half a second (Vrablik 1994). The ModSAF standard specifies that the system is running within established limits if 90 percent of the individual vehicle ticks are occurring in less than half a second (Vrablik 1994).

However, if a tick is defined as the act of passing control of the processor to a particular entity to perform an update (as is the case for ModSAF), then a problem arises when attempting to directly compare vehicle ticks from the two systems, since a tick in ModSAF is not the same as a tick in the Testbed. Therefore, it is impossible to directly compare–or even establish, for that matter–standardized benchmarking results across the two systems (at least by using vehicle ticks as the standard metric). Likewise, it would be exceedingly difficult to establish a sound conversion between a tick in ModSAF and a tick (or ticks) in the Testbed.

4.3 What, Then, Will Be Our Metric?

This is not to say, however, that vehicle ticks are not an adequate measure of the performance of the Testbed; quite the contrary. It merely means that the Testbed has a different performance threshold than does ModSAF in terms of the average vehicle tick interval. In other words, if we are to use vehicle ticks to measure performance, system degradation will become apparent at different tick rates for the two systems.

4.3.1 How Do We Measure System Degradation?

The most obvious manner in which we can measure system degradation is by observing the movements of the simulated vehicles. In the Testbed, a vehicle's route is broken down into smaller sub-routes, called waypoints (Smith 1992), which allow the vehicle to avoid the obstacles along its path. While the vehicle is moving toward its waypoint, it places update-position messages in the message queue at regular intervals. As the vehicle nears its waypoint it places a new message on the queue at such an interval that under optimum conditions, the message will be dispatched at exactly the time in which the vehicle hits the waypoint.

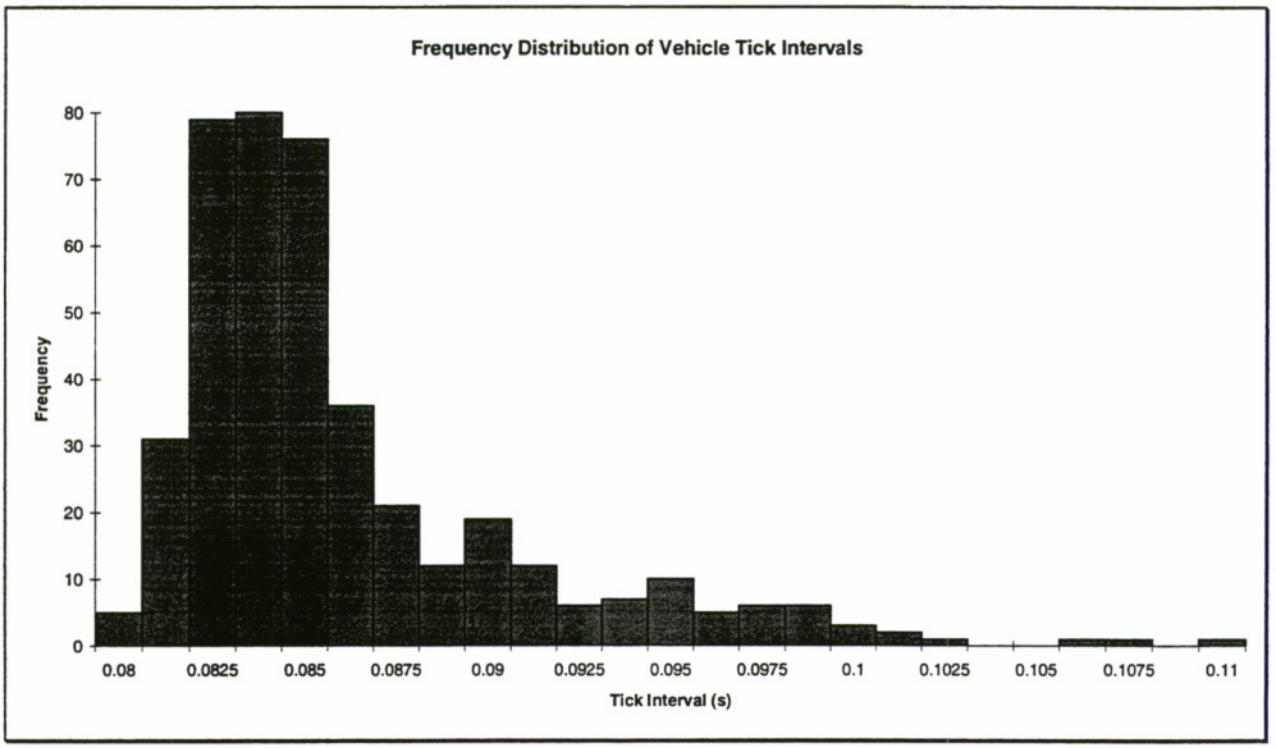


Figure 1

However, if the message is dispatched late, due to an overload on the system, the vehicle will have passed its waypoint by the time the new update occurs. This causes the vehicle to compensate by turning back–it needs to reach its waypoint before it can continue. If the system is loaded sufficiently, the vehicle will again miss its waypoint in the same manner. This degradation is marked by the vehicle endlessly circling in a futile attempt to reach its waypoint. If this "spinning" happens to any vehicle during the simulation, then the simulation has sufficiently degraded, causing unacceptable behavior (Franceschini 1993).

4.3.2 Establishment of the Performance Benchmark

With the system running under optimum conditions, tick lengths should differ only as a result of the different tasks performed during a single tick. In benchmarking ModSAF, (Vrablik 1994) states that the system is running under ideal conditions if ninety percent of the vehicle ticks occur in less than 0.5 seconds. There is statistical evidence to support a similar claim for the Testbed.

By gathering data for the average tick interval for individual vehicles, it is possible to compute a tick interval under which the system can operate that will maximize the entity load while minimizing system degradation. Figure 1 is a histogram showing the frequency distribution of vehicle tick intervals. The distribution assumes a distinctly normal quality, though it does seem a bit skewed (Freund 1979). However, we may attribute this skewness to the fact that the majority of tasks requires little processing time to complete. For instance, those tasks involving entity updates, by nature, require less time than those tasks involving line-of-sight calculations, but are also much more abundant. Therefore, if we assume that individual tick intervals generally take on a normal probability distribution, we can compute a mean and standard deviation for the average tick interval to determine the system's performance threshold (Freund 1979).

4.4 An Alternate Metric

Along with measuring the average vehicle tick interval, the Testbed tracks many other performance statistics. Among these is a measure of the amount of time that the system spends performing its tasks (as opposed to sitting idle). The Testbed reflects the executive busy time as a percentage of the total execution time. Figure 2 shows a graph of the executive busy percentage versus the number of vehicles in the simulation. The graph shows that the load on the system increases steadily as the number of vehicles increases. It reaches a point, however, where the curve changes concavity and begins to flatten. It is at this point that the integrity of the system begins to degrade. In other words, we

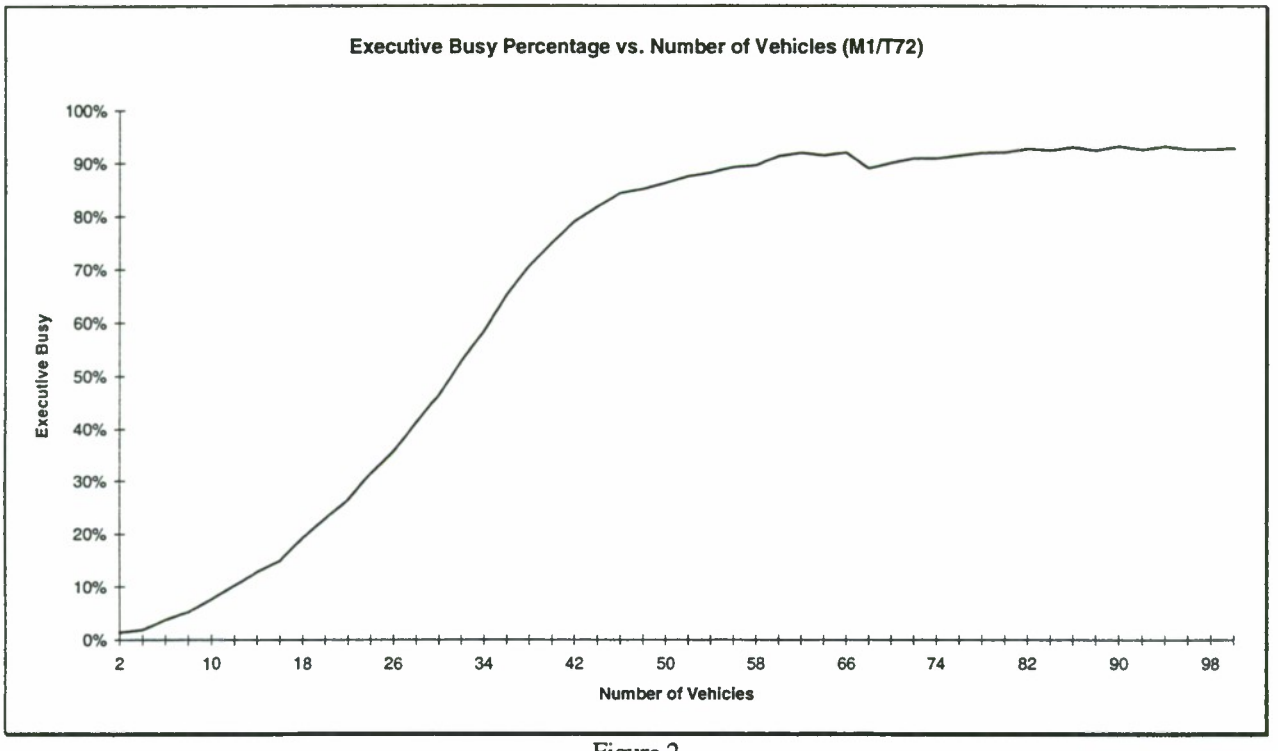


Figure 2

have reached the performance threshold. This resembles the characteristic phenomenon of network throughput times in operating system task queuing theory.

According to (Kleinrock 1976), the delay encountered by networks in sending packets remains essentially constant as network throughput increases. It reaches a threshold, however, at which point the delay grows in an unbounded fashion. In our system, this delay correlates with the average tick interval. The average vehicle tick interval remains relatively constant as the number of vehicles in the simulation increases so long as the system load has not exceeded the performance threshold. As Figure 3 shows, the total number of vehicle ticks increases steadily with the number of vehicles in the simulation–corresponding to a linear increase in the system load–but eventually reaches the threshold where the system's efficiency drops drastically. From Figure 2, we can see that the system never reaches 100 percent capacity. This merely reflects the minimum operating system overhead.

(Kleinrock 1976) shows that network delay beyond the critical threshold increases exponentially. In the Testbed, however, the placement of new messages into the message queue is entirely dependent upon the processing of messages at the front of the queue. That is, most messages trigger new messages to be placed into the queue. This dependence causes the average interval between messages (and, thus, the average tick interval) to increase in a roughly linear fashion once the system exceeds its performance threshold (see Figure 4).

5. The Benchmarking Procedure

The procedure for benchmarking the Testbed consists of six parts; the first three of which are described in (Vrablik 1994).

5.1 Part I–Two Rows of Opposing-Force Tanks

The first part of the procedure involves a scenario in which a row of blue-force tanks (M1s) engages a row of red-force tanks (T72s). All vehicles have modified damage tables that make them invincible, so they continue to move and fire throughout the entire test. The scenario begins with the two rows facing, but out of sight of one another. All of the vehicles are given permission to fire and are then told to route toward the opposing row of tanks. The benchmark lasts for five minutes, and begins after the vehicles have been moving for approximately a minute. This test is run on a completely isolated network.

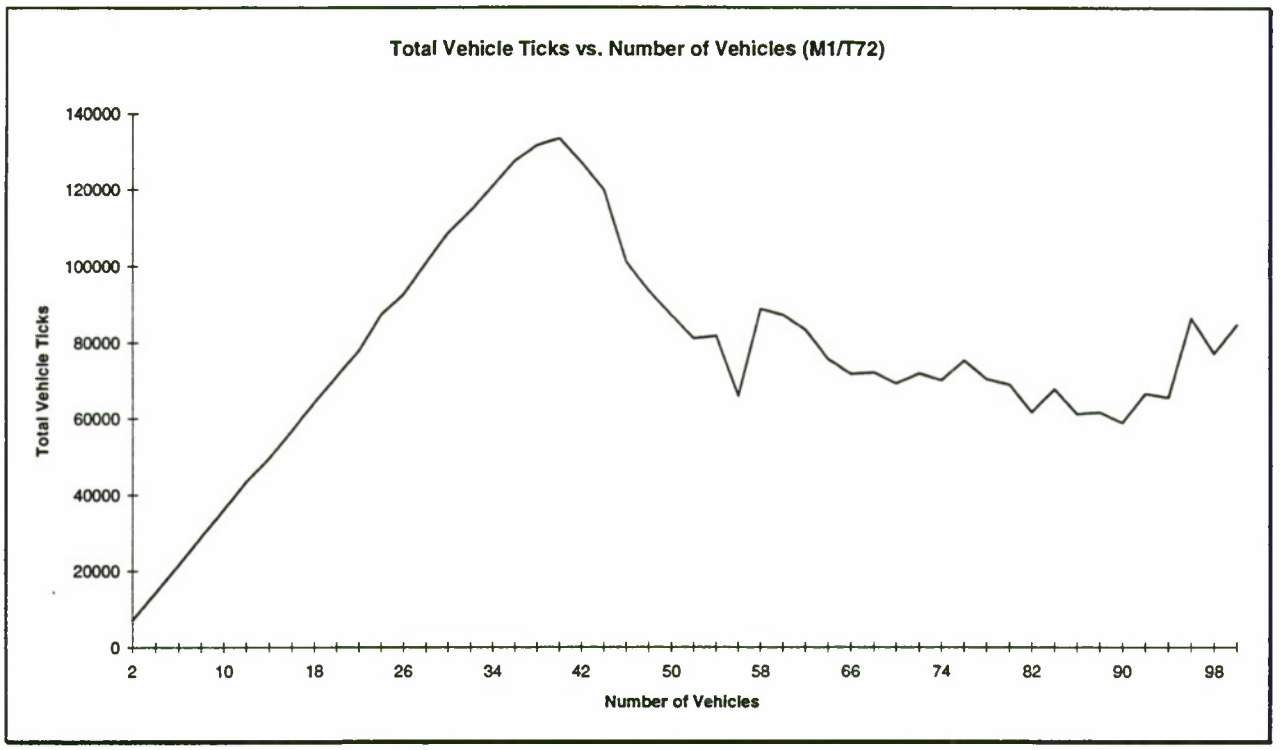


Figure 3

5.2 Part II–Remote Network Traffic I

Part II of the procedure uses the same scenario described in Part I, but with a remote network load of 800 entities created by a packet "blaster" (Vrablik 1994). The blaster is merely a utility that floods the network with Vehicle Appearance PDUs (or Entity State PDUs). For this test, the remote vehicles are generated on the same exercise ID as the local simulator allowing the interaction between local and remote entities. This test simulates the effect that a very large exercise would have on the Testbed.

5.3 Part III–Remote Network Traffic II

Part III is identical to Part II except that the remote vehicles are on a different exercise ID from the test scenario. This simulates the effect that numerous smaller exercises running on the same physical network would have on the Testbed.

5.4 Part IV–Air/Ground Interaction

Part IV is identical to Part I, except that the row of blue-force tanks has been replaced by a row of blue-force helicopters (AH64s). (Vrablik 1994) was unable to benchmark a scenario of this nature because, at the time, units of air vehicles had not yet been created for ModSAF.

5.5 Part V–Integrated Eagle/BDS-D

The Integrated Eagle/BDS-D system links a constructive simulation with a virtual simulation (Franceschini 1995). This linkage establishes three classifications for the units that it simulates: aggregated, disaggregated, and pseudo-disaggregated. Aggregated units are those units that are modeled and represented in the constructive simulation. Disaggregated units are those units whose individual entities are modeled and represented in the virtual simulation. Pseudo-disaggregated units are those units whose behaviors are modeled in the constructive simulation, but whose individual entities are represented in the virtual simulation. In the Integrated Eagle/BDS-D system, the Eagle Manager provides a key segment in the link between the constructive and virtual simulations. Essentially, it directs the flow of information between the two worlds. Thus, in order to produce a complete benchmark of the Integrated Eagle/BDS-D system, a benchmark of the Eagle Manager is necessary.

The experiment involves a demonstration scenario that the developers of the Integrated Eagle/BDS-D system have created and consists of three trials. The scenario starts with a large number of aggregated units represented in the constructive simulation. All of the aggregated units are pseudo-disaggregated, placing as

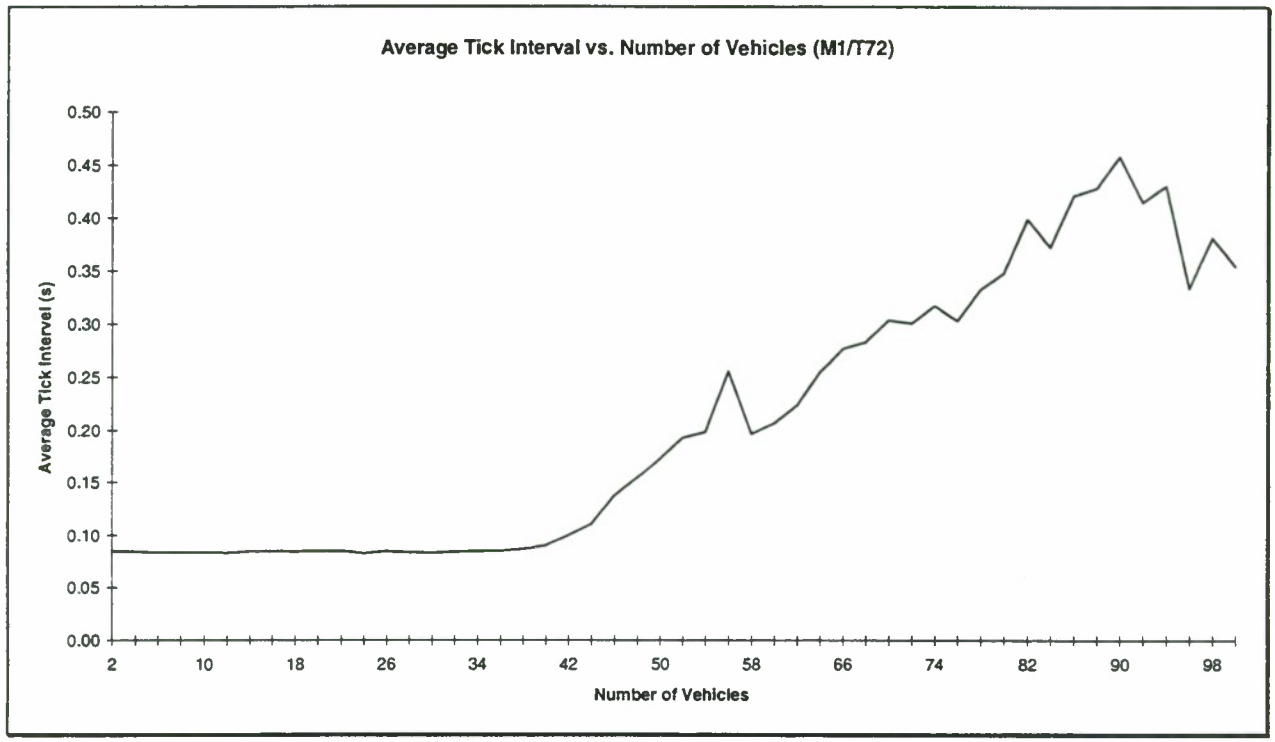


Figure 4

much a load as possible onto the Eagle Manager. Each trial then consists of fully disaggregating an increasing number of units into the virtual environment.

The Eagle Manager's job here is threefold: it shows in icon form the locations in the virtual world of the aggregated units being modeled in the constructive environment; it places the individual vehicles into the virtual world that compose the pseudo-disaggregated units being modeled in the constructive environment; and it reflects the status in the constructive world of the disaggregated units being modeled in the virtual environment.

5.6 Part VI–SIMNET versus DIS

All of the previous procedures involved testing the system in SIMNET. The Testbed (and, hence, the Integrated Eagle/BDS-D system), however, is compliant under both SIMNET and DIS. Since the underlying simulation process is identical between the SIMNET system and the DIS system (in other words, only the protocols differ), it is possible to directly compare the efficiency of the two protocols. Part VI, therefore, is identical to Part I, except that the system is running under DIS rather than SIMNET.

5.7 The Testing Environment

The Testbed was designed to run on IBM-compatible personal computers. The Integrated Eagle/BDS-D system uses Eagle as its constructive simulation, which runs on a SparcStation2. All tests involving only the Testbed were conducted on identical Dell Pentium personal computers with clock speeds of 60 MHz, and 16MB of RAM. To provide maximum performance, graphics output was turned off.

6. Experimental Results

6.1 Part I

Figure 4 shows a graph of the average interval between vehicle ticks in the Testbed. Notice that the interval remains constant (at about 0.08 seconds) for scenarios consisting of fewer than forty vehicles. Therefore, we can conclude that the system is running most efficiently when the average interval between vehicle ticks is about 0.08 seconds. This does not mean, however, that if the average tick interval is greater than 0.08 seconds, the system is running beyond its capacity. Thus we need to determine the point at which the system begins to function so inefficiently that degradation of the simulation occurs.

6.1.1 When Does System Degradation Begin?

We have already stated that one method of measuring system performance is by monitoring the behavior of

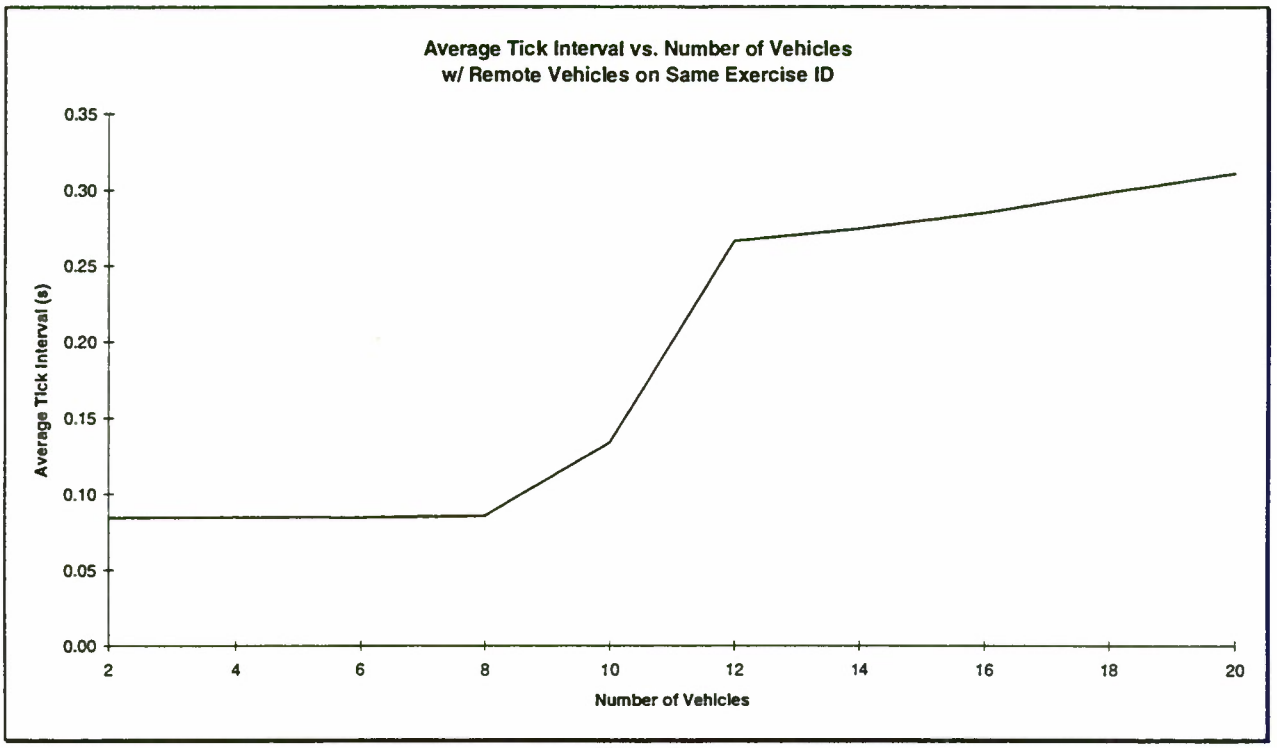


Figure 5

the simulated entities. We have observed, however, that degradation of the system begins before a noticeable decay in the behaviors of individual vehicles. As Figure 4 shows, the system is running most efficiently when the average interval between vehicle ticks is about 0.08 seconds. At higher tick intervals (say, 0.10 seconds), the Testbed may still appear to perform reasonably well, but system degradation has already begun. The question remains, then, "At what point does sufficient degradation occur in the system as to affect the behaviors of individual vehicles?" The graph of the total vehicle ticks shown in Figure 2 increases steadily until it takes a dramatic nose-dive after forty vehicles. This shows that under the conditions created by the test scenario, the system runs without degradation with a load of up to forty vehicles. After forty vehicles, degradation begins. We can now use the forty-vehicle scenario to determine the Testbed's performance threshold by establishing a range for the average tick interval based on those individual intervals measured for the forty-vehicle scenario.

Given the normal distribution of vehicle tick intervals, we can calculate the mean and standard deviation for the average tick interval using the forty-vehicle scenario as the standard. It is in this manner that we have determined that the Testbed is running with no system degradation when the average vehicle tick interval is 0.0922 seconds. More precisely, we can say with ninety percent confidence that the Testbed is running with no system degradation when the average vehicle tick interval is between 0.0840 and 0.1004 seconds (Freund 1979).

6.2 Parts II, III, and IV

Figure 5 shows the average tick interval for vehicles in the scenario where the system is under a network load of 800 remote entities on the same exercise ID. It is clear that the remote network load places an extra burden on the system, since at the 0.0922-second tick interval threshold, the system is at capacity with a local load of only eight vehicles.

Figure 6 shows the average tick interval for vehicles in the scenario where the system is under the same network load, but with the entities on a different exercise ID. The threshold in this case appears to be at about thirty-six vehicles. Therefore, the presence of a large number of exercises running on the same physical network should not critically affect the performance of each exercise.

Figure 7 shows the average tick interval for vehicles in the scenario involving air/ground interaction. Notice that the Testbed reaches its capacity with fewer vehicles. The primary weapon that helicopters use in

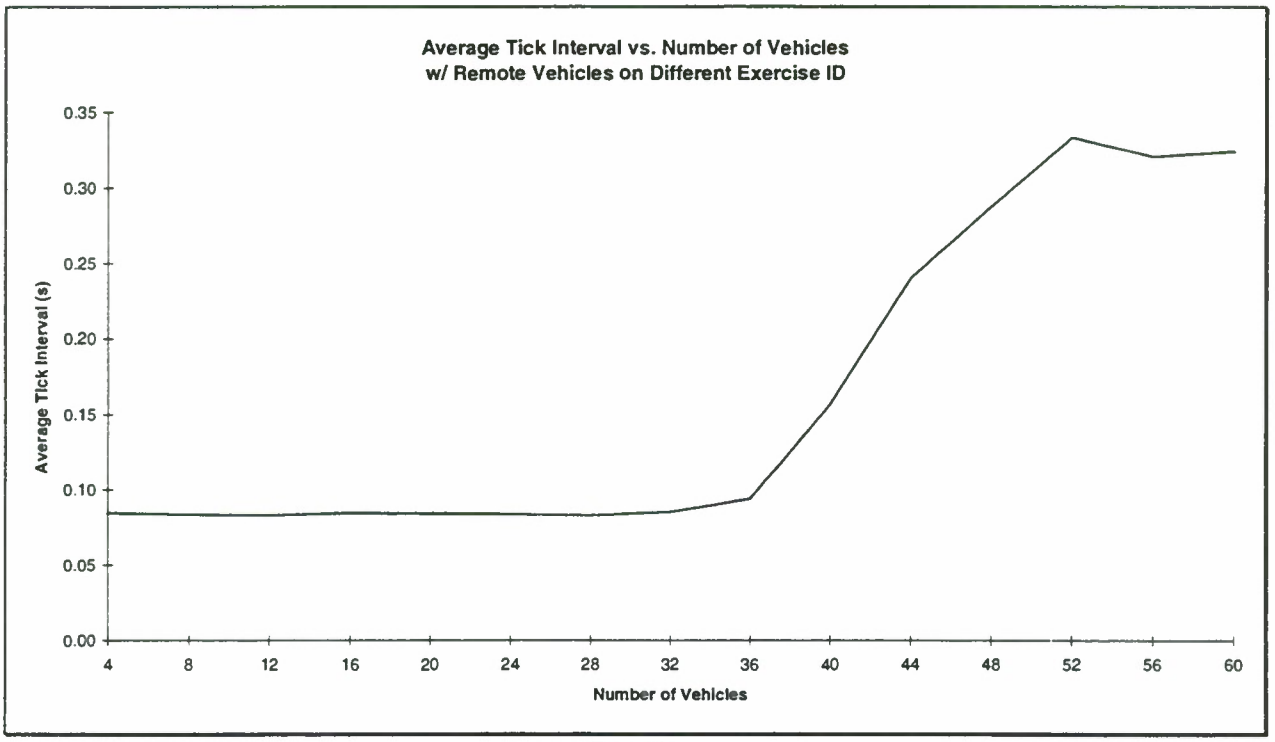


Figure 6

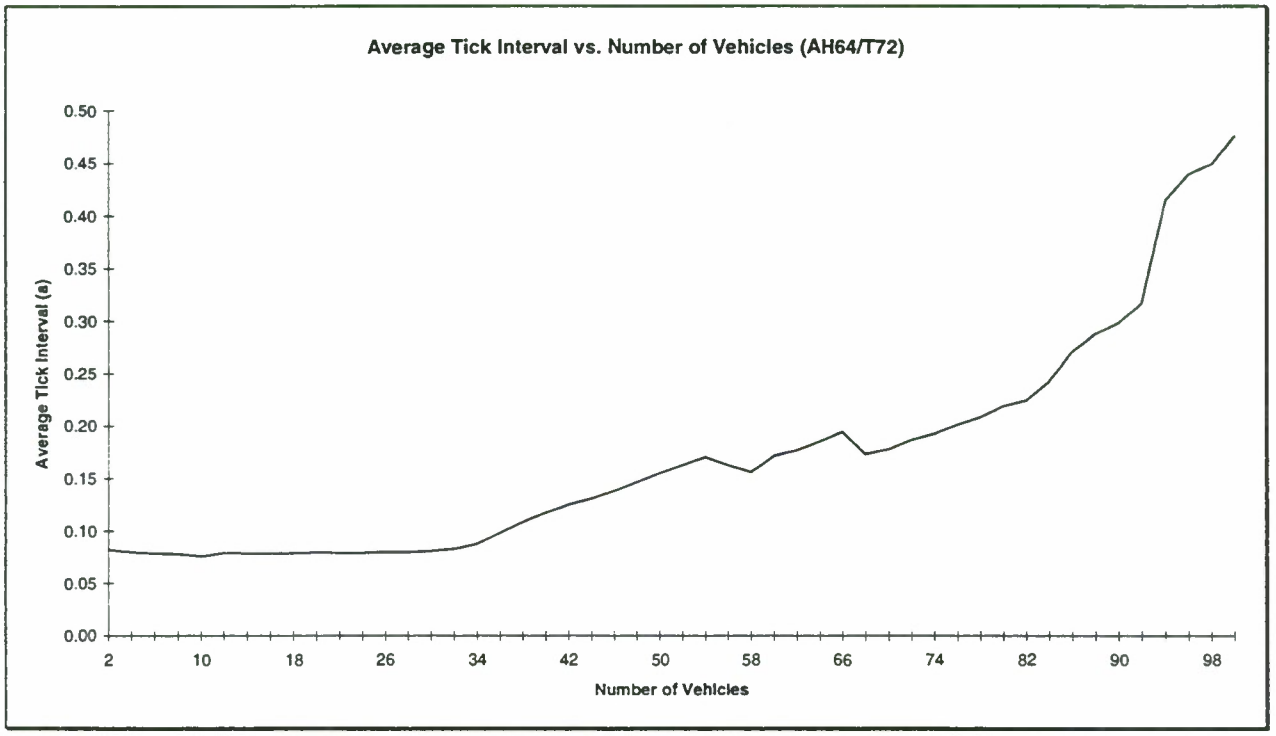


Figure 7

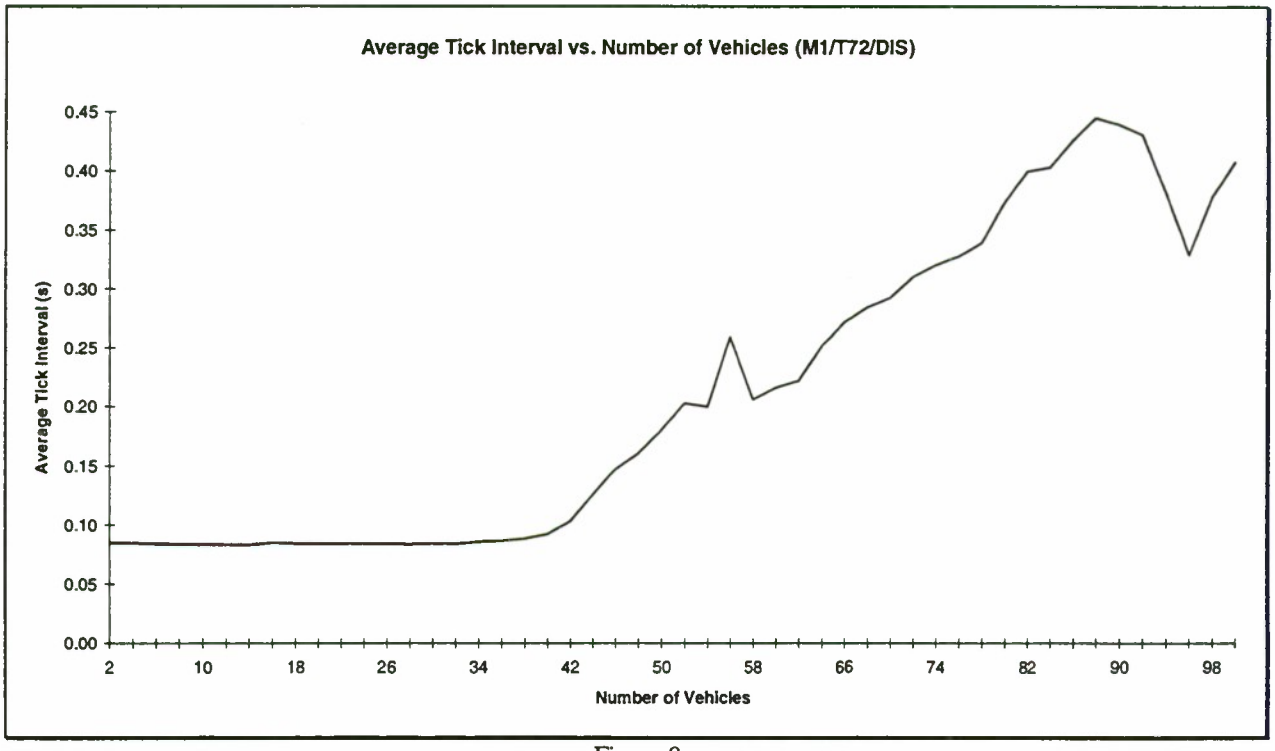


Figure 8

engaging tanks is the guided missile, which, unlike other forms of fire, is itself represented as an individual entity in the virtual environment. Therefore, we can attribute the smaller capacity to the additional overhead that the missile flyouts generate.

6.3 Part V

The Eagle Manager's operating characteristics are very different from those of the simulation host. In the linkage between the constructive and virtual simulations, the Eagle Manager acts primarily as a packet blaster, placing Vehicle Appearance PDUs (or Entity State PDUs, depending on the protocol) on the network to represent pseudo-disaggregated vehicles in the virtual environment. Otherwise, the Eagle Manager remains mostly idle, waiting to route the occasional message from one side to the other. When it does receive a message, such as a request for disaggregation, the Eagle Manager operates at or near capacity through the completion of the task at hand. It then returns to its somewhat idle state. However, due to the non-preemptive nature of the Eagle Manager, each task finishes before the next one starts. Therefore, the only way to overload the Eagle Manager is to attempt to pseudo-disaggregate so many vehicles that it simply can not place Vehicle Appearance PDUs on the network fast enough. To this extent, the Eagle Manager is limited by the capacity of its network interface hardware. We were therefore unable to determine a limit on the capacity of the Eagle Manager.

6.4 Part VI

Figure 8 shows the average vehicle tick interval for the tank scenario under the DIS protocol. From these results, it appears that there is very little difference (if any) between the two protocols. DIS, however, does support a much wider variety of PDUs than does SIMNET. Therefore, these results may be skewed since the test scenario did not require the utilization of those PDUs that are specific to DIS.

7. Conclusions

7.1 The Test Conditions–A Caveat

It is important to note that the test scenarios were conducted with altered damage tables for those entities in the simulation. By making the vehicles invincible, the test procedure produces a worse-than-worst-case scenario in which all of the vehicles are continuously routing, targeting, and firing–yet never dying.

7.2 Impact of Terrain on the Benchmark

It is also important to note the limitations that the terrain imposes on such a test (Vrablik 1994). If the terrain were completely flat and contained no obstacles,

every vehicle would be able to see every other vehicle, thus creating a distorted view of the capacity of the system. Likewise, if the terrain were extremely rough, a similar distorted view might result. Therefore, the vehicle tick interval seems a trustworthy measure of the system load because it removes all of these factors from the equation. The bottom line becomes: The system is running within its limits if the average interval between vehicle ticks is within the determined range.

7.3 Limitations of the Executive Busy Percentage

The use of the executive busy percentage as a measure of performance can also be disputed. The executive busy percentage measures the amount of time that the executive spends processing its tasks. It does this by determining the difference between the time at which the task was dispatched and the time at which the task was finished. However, given the granularity of the timer, a task might finish before any measurable amount of time has passed.

Obviously, the purpose of benchmarking such a system is not merely to determine how many entities it can support. Its primary goal is to provide a standard by which we can measure future performance.

7.4 Future Work

IST has not yet integrated the measurement of the vehicle ticks into the Testbed, mainly because of the abundance of performance data that it already gathers. With the possible exception of the executive busy percentage, however, the average vehicle tick interval seems to be the most accurate measure of the system's performance. Therefore, the future developers of the Testbed may decide to implement the vehicle tick measurement into the system.

When the benchmarking option is specified from the command line in ModSAF, not only does the system tally vehicle ticks, but it also halts the simulation as soon as it determines that it is running beyond its capacity. In any case, it would be very helpful for the system to at least warn of possible spurious behaviors resulting from an impending system overload.

The accuracy of the executive busy percentage as a measure of system performance is questionable due to the granularity of the timer involved in measuring the executive busy percentage. It is apparent, though, that a finer timing mechanism would produce more accurate results.

Lastly, more work is needed in developing a benchmark for a constructive+virtual linkage, because of the complex nature of this type of system.

8. Acknowledgment

This research was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command as part of the Integrated Eagle/BDS-D Project, contract number N61339-92-K-0002. That support is gratefully acknowledged.

Special acknowledgment goes to Michael A. Craft and Robert D. Russell, who provided invaluable guidance from their knowledge of operating systems.

9. References

Franceschini, Robert W., Watkins, Jon E., Parra, Fernando R., McCulley, James E., Lautenschlager, Jennifer A., Jackson, Lance A., Nanda, Sanjeeb (1993). "SAFDI Support Manual", *Technical Report IST-TR-93-24*, Institute for Simulation and Training, University of Central Florida.

Franceschini, Robert W. (1995). "Integrated Eagle/BDS-D: A Status Report", *Proceedings on the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, May 9-11, 1995.

Freund, John E. (1979). *Modern Elementary Statistics*, Fifth Edition, Prentice Hall.

Karr, Clark R., and Root, Eric D. (1994). "Integrating Aggregate and Vehicle Level Simulations", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando, Florida, May 4-6, 1994, pp. 425-435.

Kleinrock, Leonard (1976). *Queueing Systems–Volume II: Computer Applications*, John Wiley and Sons.

ModSAF Programmer's Reference Manual, (1994). Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1994.

Smith, Scott H., Karr, Clark R., Petty, Mikel D., Franceschini, Robert W., Wood, Douglas D., Watkins, Jon E., and Campbell, Charles E. (1992). "The IST Semi-Automated Forces Testbed", *Technical Report IST-TR-92-7*, Institute for Simulation and Training, University of Central Florida.

WATCOM C/C++ Programmer's Guide, (1994). WATCOM International Corporation,

Waterloo, Ontario, Canada.

Vrablik, Rob, and Richardson, Wendy (1994). "Benchmarking and Optimization of ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando, Florida, May 4-6 1994.

10. Biographies

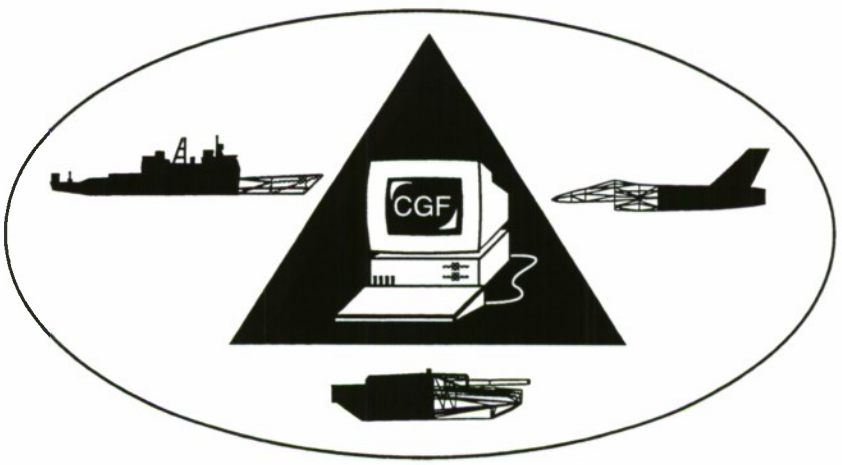
Stephen A. Schricker is a Software Engineer on the Integrated Eagle/BDS-D project at the Institute for Simulation and Training. He has earned a B. S. in Computer Science from the University of Central Florida. His research interests are in the area of simulation.

Tracy R. Tolley is a Graduate Research Assistant on the Integrated Eagle/BDS-D project at the Institute for Simulation and Training. She has earned a B. S. in Mathematics from the University of Central Florida, and is currently pursuing an M. S. in Computer Science from UCF. Her research interests are in the area of simulation.

Robert W. Franceschini is a Principal Investigator at the Institute for Simulation and Training. He currently leads the Integrated Eagle/BDS-D project at IST. Mr. Franceschini has earned a B. S. in Computer Science from the University of Central Florida; he is currently pursuing an M. S. in Computer Science from UCF. His research interests are in the areas of simulation, graph theory, and computational geometry.

Session 9b: Dismounted Infantry

Howard, Hughes Research Laboratories
Mastroianni, U.S. Army , Natick RD&E Center
McIntyre, Simulation Technology Inc.
Middleton, Simulation Technology Inc.


CGF

Individual Combatant Development in ModSAF

M. D. Howard[1], B. Hoff, and D. Y. Tseng
Hughes Research Laboratories
Malibu, CA 90265

1. Abstract

Individual Combatant (IC) capabilities have been developed in ModSAF as part of the Marine Corps Semi-Automated Forces requirements for the LeatherNet project and STOW '97. The initial development effort has been focused on modifying and enhancing ModSAF to enable the inclusion of the Marine Corps IC at the lowest echelon levels: Fire Teams and Rifle Squads. Additionally, three special teams (60mm Mortar team, Machine Gun team, and Assault team) have been created. Early development concentrated on basic functions and basic movements of the ICs, which are building blocks for more complex tactics and maneuvers. These results have been incorporated into Build One of the Marine Corps LeatherNet project.

The modification to, and enhancement of, ModSAF to create the appropriate Marine IC functionality will be described. This was not a trivial undertaking because of ModSAF's orientation toward Army armor. Also, the integration of IC formations and movements with planning techniques which utilize terrain features for cover and concealment [presented in more detail in a companion paper by Hoff, et al] will be discussed.

2. Introduction

2.1 Previous SAFOR work

Hughes Research Labs (HRL) first applied autonomous vehicle navigation ideas (Mitchell, et. al, 1987) to the problem of tank platoon and company movement with formation maintenance in the SIMNET Semi-Automated FORces (SAFOR) project in 1990. A *concurrent-control* scheme for driving SAFOR tanks balanced the competing goals of avoiding obstacles, maintaining a formation, and following an assigned route. This control scheme showed a measured improvement over the finite state machine approach used in SAFOR (Harmon et al, 1990). We also applied concurrent-control to the gunner and company commander. In particular, our concurrent control commander (Harmon, et al, 1994) was able to balance competing objectives to make decisions about breaking formation to respond to incoming enemy fire and sending orders to subordinate commanders by means of PDUs. In phase two of this project we applied case-based reasoning and concurrent control to the air vehicles in the WISSARD IFOR program (Keirsey *et al*, 1994), which was built on top of ModSAF A.

2.2 The Need for USMC SAF

The goal of HRL's work in the current LeatherNet project is to develop the Individual Combatant (IC) capabilities for the Marine Corps component of STOW97. Early versions (A and B) of ModSAF included models of air vehicles for the WISSARD IFOR program, and the 1.X versions have concentrated on Army armor units. IC capabilities required for Marine Corps simulation have not been sufficiently developed in ModSAF. A ModSAF Dismounted Infantry (DI) is basically a slow, vulnerable tank with a rifle, that can mount and dismount Infantry Fighting Vehicles (IFVs). This is not an unreasonable way to simulate an IC given ModSAF's existing code library and the need to conserve CPU cycles. For instance, the hull model in ModSAF simulates variations in speed with soil type and the turret / loader / gunner model simulates ammunition supplies, loading time, targeting/tracking and shooting. These models can be applied to ICs with appropriate parameter changes.

Some existing tasks such as VMove, which controls vehicle movement, can be reparameterized to get a quick start of the behavior, but ultimately some deeper aspects of the behavior need to be addressed. For example, a tank considers a tree canopy to be a penetrable obstacle; that is, it will detour around the canopy unless specifically tasked to go inside. An IC is probably attracted to a tree canopy as it affords

[1]To whom correspondence should be addressed via internet: howard@isl.hrl.hac.com

concealment without significantly slowing forward progress. Attraction to an area of concealment is not well developed in ModSAF; section 3.4 discusses our approach. Furthermore, there are tactical considerations that might make an IC stay out of a tree canopy, including the need to stay in visual contact for hand signals not used by the tanks. Some of these higher level behaviors may eventually require the more sophisticated blending of lower behaviors called concurrent control.

There is no model of "suppression" (direct or indirect fires brought to bear on an enemy to prevent effective fire on friendly forces) in ModSAF, or of a response to suppression. We will need to model the way that suppressive incoming fire affects a soldier's firing rate and targeting accuracy. The ability to signal other units by means of smoke and flares is not present in ModSAF as of version 1.4. Finally, Marine Corps echelons, formations, techniques of movement and tactics are not represented in ModSAF.

In the next section, our enhancements to ModSAF for the LeatherNet Build One demonstration (six months into the program) is described in some detail. To date we have addressed a subset of needs mentioned above. Our successful Build One demo is described in section 4, "Results".

3. ModSAF Enhancements

3.1 Integration Choices

The following discussion will be more understandable if we define two terms to describe how a developer's code is integrated with ModSAF: *loose* and *tight*. A *loose* integration implies that the developer's code is a separate code module, with few links into ModSAF. An extreme example of this might be a module that runs in a different process, using a low bandwidth socket or shared memory interface. A more moderate example is the way SOAR was integrated, running in the same process but bypassing ModSAF's movemap for vehicle control and providing its own user interface (Schwamb *et al*, 1994). The advantages of a loose integration with ModSAF include easier porting to new versions of ModSAF and avoidance of much of the ModSAF learning curve. The drawback for the developer is the likelihood that certain ModSAF code will need to be duplicated. As a result, ModSAF's already complex code can become duplicative and disorganized.

A *tight* integration uses standard ModSAF mechanisms as much as possible. In this method, the developer's code adds functionality to ModSAF by making changes as subtly as possible. Developers desiring to embed their control mechanisms in ModSAF in this way face a steep learning curve because a deep understanding of ModSAF is required. It is riskier and more expensive, at least in the short term, to do this.

At the beginning of HRL's involvement in the LeatherNet program, the 1.0 version of ModSAF which was then available did not simulate two-echelon-deep units like an Army company. We were aware that company level capabilities were being developed and would soon be available, but our schedule was too tight to wait. In order to show an early capability in the LeatherNet program, we chose to use some basic movement, formation, and echelon logic we had developed for another program, and to *loosely* integrate it with ModSAF. In other words, a Marine Corps entity would run our concurrent control code for route planning, obstacle avoidance, and formation keeping, instead of running the ModSAF movemap planning code.

When ModSAF 1.2 was delivered and contained company level capabilities, some of our code became redundant. We opted to use the new ModSAF echelon and formation code and to use ModSAF's movemap for movement, thereby shifting to a tight integration. The next section describes our resulting tight integration with ModSAF 1.2 in some detail.

3.2 Creation of Marine Corps Entities

3.2.1 Marine Corps Rifle Squad

Our first job was to create, in ModSAF, a Marine Corps rifle squad. In ModSAF, a tank is an atomic entity which can move, shoot, and communicate. ModSAF models two echelons for Army armor: a platoon composed of tanks, and a company composed of platoons. For simulation purposes, a Marine Corps squad has a similar two echelon structure: a fireteam composed of Individual Combatants (ICs), and a rifle squad composed of fireteams. This is important because the echelon organization determines how a behavior task is structured in ModSAF, and we were able to obtain early results in Marine Corps modeling by re-coding some ModSAF company tasks for the Marine squad.

For example, a ModSAF company task such as "UCMarch" is a meta-task that spawns and controls platoon tasks such as "UTraveling", which in turn

spawns individual vehicle tasks such as "VMove". But ModSAF's company unit tasks could only be run by units that were called "Company." It was necessary to extend them so they could also be used, with some revisions, on Marine rifle squads. This involved not only changes in parameters such as formation spacings and speeds, but also changes in some of the finite state machine logic used to implement these behaviors, written with Army doctrine in mind. Some examples of this are discussed below in section 3.3, "Formations and Basic Movement."

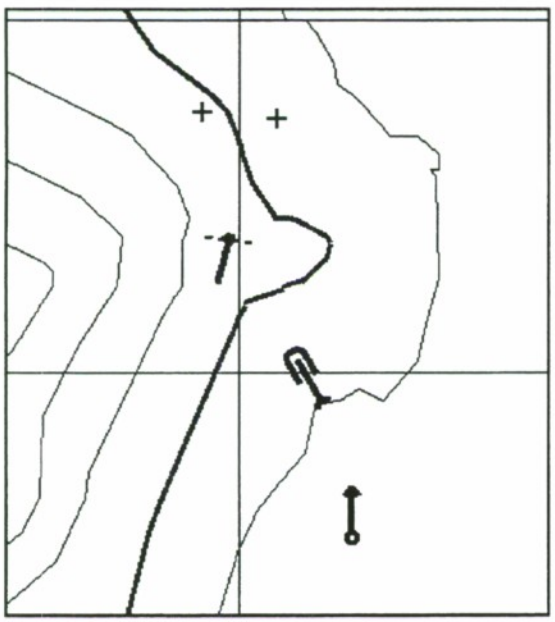


Figure 1. A machine gun team (upper left) is sitting at its base of fire on a ridge. A mortar team (lower right) and a machine gun team are shown traveling nearby. Each team is portrayed by a single icon, which represents the team's weapon. (The grid shown is 100 meters).

3.2.2 Special Weapons Teams

Three special weapons teams were needed in the Build One context: an assault team armed with a Mk153 SMAW, a M224 60mm mortar team, and a machine gun team armed with a M240G machine gun For Build One, it was only necessary to simulate the functionality of each of these teams. In this case we used a simple entity that could move to a location, simulate setup time, and then orient itself on a target. Setting up the weapon was simulated by having the team icon wait for a specified after arriving near the base of fire before moving into position.

3.2.4 Marine Corps Iconic Display

The standard ModSAF icon view shows at a glance which entities are ModSAF Dismounted Infantry (DI) and which are vehicles. But in order to debug the [simulated] Marine formations and enable [real] Marines to VV&A our work, we had to display more information for each entity in the rifle squad. Military symbols for Marine Individual Combatants (ICs) denote such information; they graphically differentiate each squad member (squad leader, fireteam leader, rifleman, etc.) by portraying its *role* in the squad. ModSAF does have a concept of role: the job of unit leader can be passed from one entity to another in a defined order, if the entity currently performing that job is killed. However, icons displayed on ModSAF's plan view represent only entity type. Using the "View As..." menu item, the user can choose to see either a "picture" or an "icon" of the entities. But the information presented in either case is basically the same.

The only method ModSAF provides to override the default and attach a particular icon to an entity is to list the entity and its icon individually in a reader file. To use this method, it was necessary to create five Marine Corps entities (squad leader, fireteam leader, machine gunner, machine gunner's assistant and rifleman), one for each role in the squad. The appropriate military icon was attached to each individual in the squad. Role icons can now be displayed by selecting "Vehicle Icons" from the "View As..." menu in the ModSAF plan view display. An example of this view is shown in figure 2.

Though the roles of the squad members are now visible, the mechanism we used has two drawbacks for the long term which will have to be addressed in the future. First, though several members of a fireteam carry the same weapon, it was necessary to make them different entity types just for the purpose of displaying an icon. This is undesirable not just because it is unnecessarily duplicative, but because ModSAF hard-codes the maximum number of entities in a unit to be four and makes several program loops and arrays dependent on that number (there are five roles in a squad). Most importantly, in the Marine rifle squad, it is not just the leader's role that can migrate. For example, if the Automatic Rifleman (who carries an M249 Squad Automatic Weapon or SAW) is killed, his assistant picks up the SAW and becomes the Automatic Rifleman. ModSAF only allows us to attach an icon to an entity in a reader file, at startup time, so we cannot change the role icon to simulate this migration of roles until this software design is enhanced.

3.3 Formations and Basic Movement

3.3.1 Squad and Team formations

Our extension to basic ModSAF formations for the Marine Corps consists of two parts: a mechanism for specifying a leader's position within the squad, and a mechanism for creating different team formations within a squad formation.

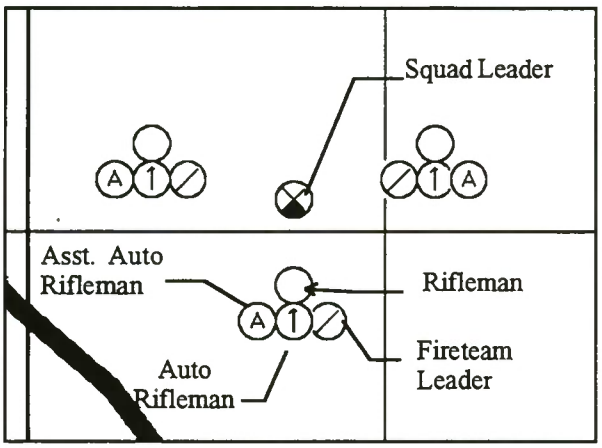


Figure 2. A squad in vee formation. Note that some fireteam formations are mirror images of each other. The individual combatants (ICs) are displayed on the ModSAF plan view display using military icons that identify their roles. The text labels in the figure were added to the figure for explanation. (Squad is shown on 100m grid).

3.3.1.1 Leader's Position within Squad

In ModSAF, formation specifications for a company only apply to subordinate units (the platoons). Leaders (XO and CO) are, by convention, situated behind the formation. In fact, ModSAF was designed so that leaders of a company cannot be put anywhere else when the company is created. This is another case in which ModSAF's basic design was not general enough to easily accommodate a different military service. Because the leader of a Marine Corps squad is usually closer to the center of the squad for better control, we added to ModSAF the capability to flexibly specify the position of a leader within the squad or company formation.

Our solution works within ModSAF's formation specification mechanism, so the squad leader's position is specified using the standard formation syntax. One possible future need that we have not yet addressed is to allow a squad leader to move around to different positions within the squad dynamically, depending on tactical situation as well as formation. We hope to accomplish this within the ModSAF framework as well.

3.3.1.2 Mixing Subformations

Using the new formation specification capability, a set of new formations was generated. The following table lists the Marine Corps formations for fireteams and rifle squads that have been implemented for Build One. The designation "(R/L)" indicates that the formation can be oriented differently (right or left) depending on circumstance or position within a larger formation.

RIFLE SQUAD	FIRETEAM
Tactical Column	Tactical Column
Online	Online
Wedge	Wedge (R/L)
Vee	Vee (R/L)
Echelon (R/L)	Echelon (R/L)
Rear Point	(nominal)
	Skirmisher's (R/L)

Table 1. Marine Corps formations simulated for Build One demo.

Several Marine squad formations are composed of mixed team formations. For example, a rifle squad rear point formation, shown in figure 3 below, consists of fireteams in echelon right, echelon left, and wedge subformations (indicated by "nominal" designation in the table above). A squad vee (figure 2) has each fireteam in a vee, but each wedge is oriented differently. Standard ModSAF did not allow mixed subformations. It allows the user to specify only a single subformation type for each formation. Our extension to ModSAF allows the user to choose the formation and have the correct subformations automatically filled in. But the standard ModSAF functionality is still available: if the user specifies a certain subformation, that subformation is used for all sub-units.

3.3.2 Basic Movement

In standard ModSAF, though a leader is located behind its company when created, it functionally joins one of the platoons when the company is tasked to move. Each movement task recreates the company (or squad) formation when specifying the subordinates' routes. The task uses its own rules, including spacing parameters.

With the close formation spacing required for Marines (on the order of 10-20m between ICs vs. 50-100m between tanks), excessive collisions occur between members of the squad traveling along the same route

(*e.g.*, when in a column formation). This is because ModSAF gives each member of the team a private route, and plans its speed in advance based on the maximum speed an entity can travel on the terrain. However, the entity constantly adjusts its speed as it travels, and it can get behind its planned position. If the entity behind is following closely and doesn't anticipate the speed change, a collision can occur. Then both do a collision recovery, which involves recoiling, and that causes more bumping and more delay. This low frequency predictive approach to collision avoidance is a reasonable compromise with the need to save cpu cycles. But in complex microterrain, team members are constantly changing their speed and the problem is magnified. We will have to add some stronger station-keeping and collision avoidance to ModSAF especially to deal with column formations on the 5 to 10 meter spacings needed for ICs. We have addressed this problem in earlier work (Harmon et al, 1990), using our concurrent control mechanisms. We are still studying the problem in ModSAF, and may use concurrent control in a future build.

3.4 Advanced Movement Using Terrain for Cover and Concealment

ModSAF has a primitive capability to find covered and concealed (C&C) positions for stopping, but it had no way to use cover while moving until the grid-based C&C route planner in version 1.4. We have added an ability to intelligently plan routes so they take advantage of nearby cover and concealment. A unit's route is passed into a path planning module, where it is altered to use nearby regions of cover. This "weighted-regions" path uses an optimal path-planning algorithm based on a polygonal representation of terrain features, developed at SUNY Stonybrook. Because the weighted regions approach can deal directly with the Triangulated Irregular Network (TINs) of the microterrain, this approach is potentially faster than grid based approaches such as that used in ModSAF 1.4, and the result is not prone to "digitization bias". A companion paper (Hoff et al, 1995) in this conference presents the weighted regions path planning approach.

4. Results and Future Work

The Build One demo

The Build One scenario developed for us by our Marine SMEs involved an assault on an enemy occupying two trenches in Range 400. Figure 4 shows the ModSAF plan view display set up for the Build One scenario, which is as follows:

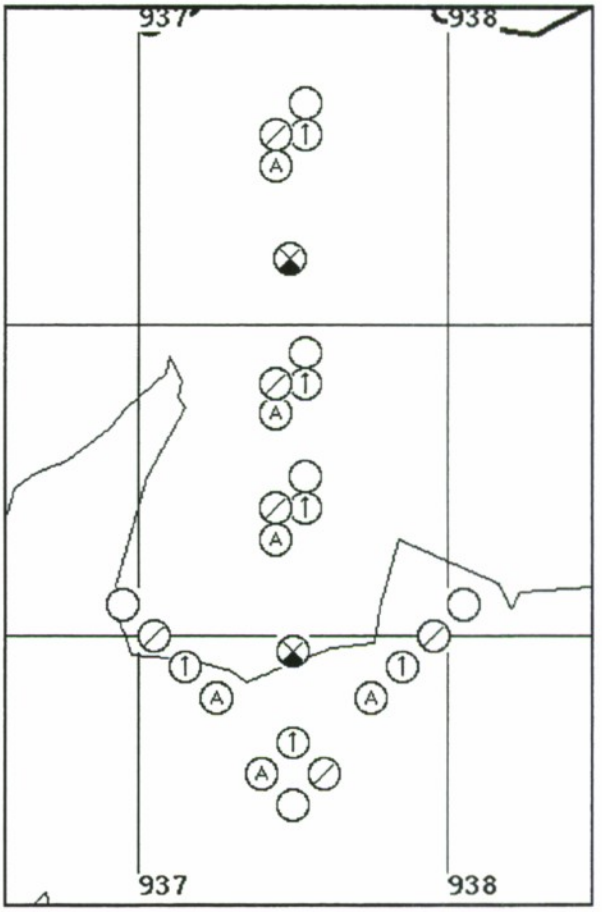


Figure 3. Trailing end of a platoon column moving north (toward the top of the page), showing 2 squads on a 100m grid. The squad in front is in a tactical column, and the rear squad is in rear point formation.

The enemy has established a base of fire at a Security Position near the center of the range, and his main force is dug into a long trench at the north end of the range, Objective A (figure 5). The terrain is a desert valley surrounded on North, East and West by mountains. Friendly forces, located at Assembly Area AA in the south end of the range, consist of a squad with attachments. The three attached special weapons teams are an assault team armed with a SMAW, a machine gun team with Mk 153 machine guns, and a mortar team with a 60mm mortar. The plan, developed and input by a human platoon commander, is to send the mortar team NE to CP3, where it can destroy the Security Position. Meanwhile the squad

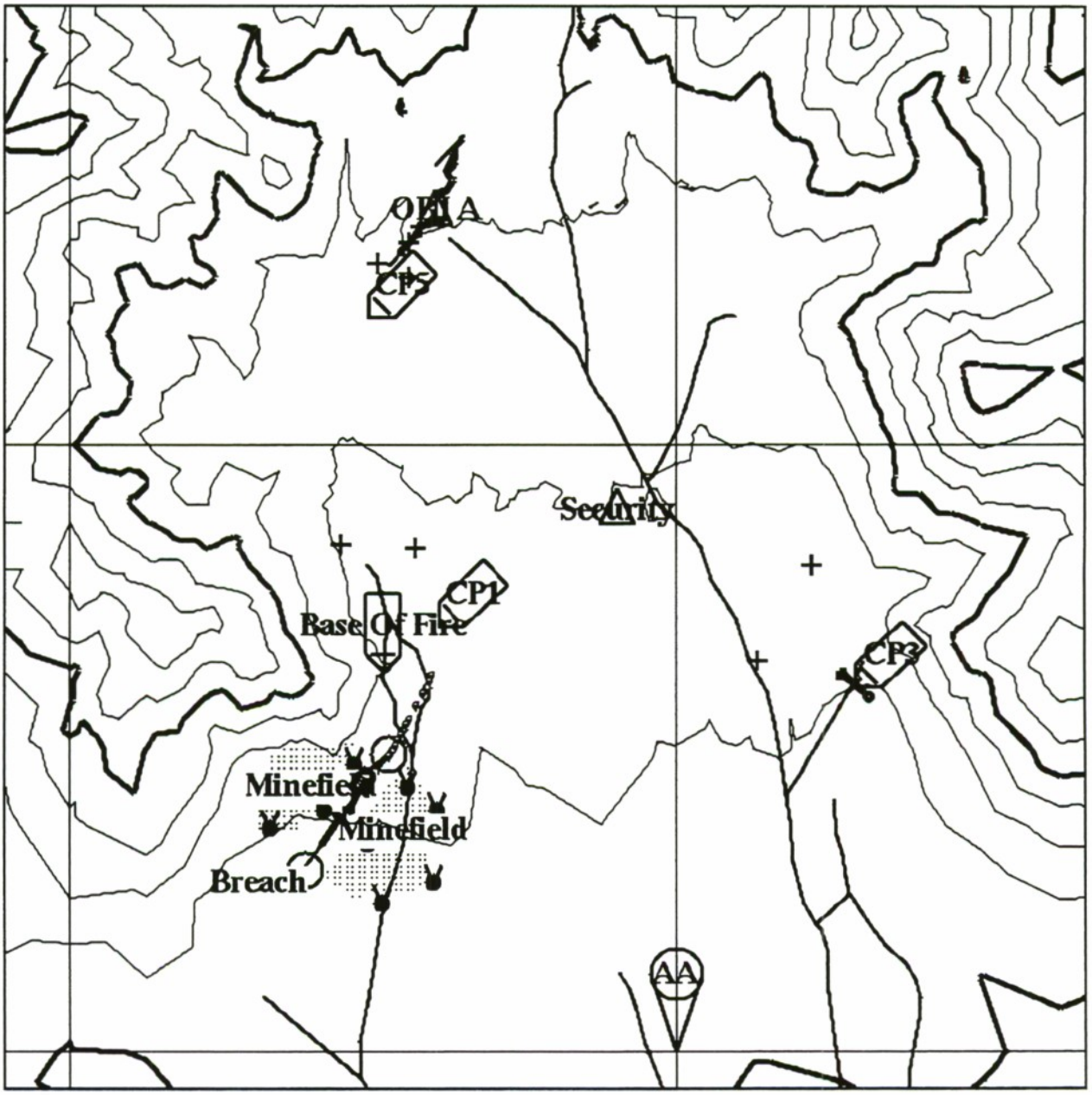


Figure 4. Build One Scenario. Squad and special teams begin at assembly area AA at bottom of map, south of the area of operations. Bases of fire are set up just north of minefield to West and at CP3 to East. Security Position target is in center, and final Objective A trench is North. Image depicts mortar team set up at CP3, oriented on Security Position, while rifle squad is just emerging from breach through minefield, with machine gun team and assault team in trace.

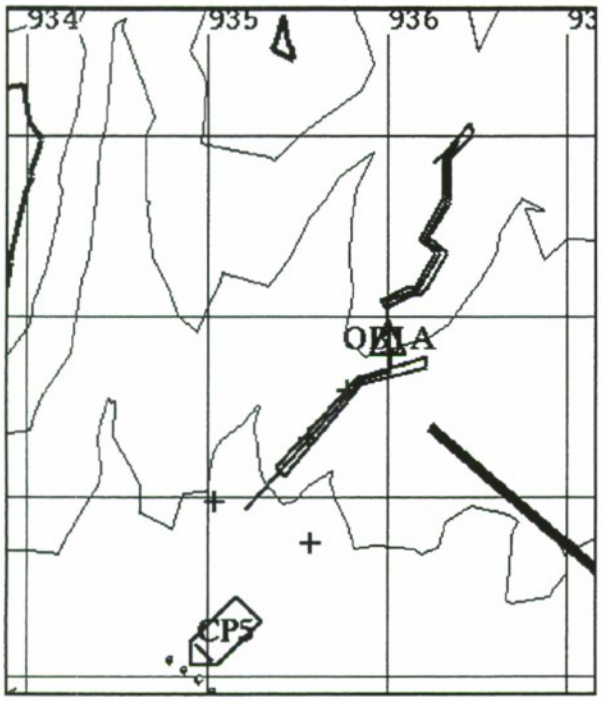


Figure 5. Squad is just arriving at final assault position CP5 at Objective A trench. (Grid is 100m.)

moves NW, breaches the Minefield, and moves North toward the Objective A trench. The other two special teams trace the squad.

After breaching the minefield, the machine gun team breaks off to establish a base of fire on a hill (marked "Base of Fire), and suppresses Objective A. The squad, with assault team still in trace, maneuvers through a series of WADIs (desert washes that provide cover and concealment), until it reaches the final assault position CP5 on the right flank of the enemy in Objective A. At this point, the machine gun team to reorients its base of fire to the eastern end of Objective A, and the squad and assault team assault the objective from the western end.

The overall goal of our work is to develop the Marine Corps component of the STOW97 exercise. Various builds in the program are working incrementally toward that goal. Our Build One demo was successfully completed in December 1994, and we have begun to work on the Build Two demo. For Build Two, planned for late August 1995, more complex behaviors will be implemented for the ICs, and engagements between OPFOR and BLUFOR will take place. The precise scenario for Build Two is still under consideration.

5. Acknowledgments

This work was supported in part by ARPA contract DAAE07-92-C-R007, contracted through NRaD. We gratefully acknowledge the guidance and support of LCDR Peggy Feldmann and LT Jeff Clarkson in the performance of this program. Mac Brewer's advice and feedback kept us aware of USMC needs. Knowledge Acquisition of Marine Corps tactics and doctrine for the LeatherNet program has been provided by BMH.

6. References

Harmon, S.Y, Yang, S. C., Howard, M. D, and Tseng, D. Y. (1990), "A Behavior-Based SAFOR and its Preliminary Evaluation", in *Proceedings of U. S. Army PM Trade 2nd Behavioral Representation and Computer Generated Forces Symposium*.

Harmon, S.Y, Yang, S. C. and Tseng, D. Y. (1994), "Command and Control Simulation for Computer Generated Forces", in *Proceedings of the Fourth Conference on Behavioral Representation and Computer Generated Forces.*

Keirsey, D. M., Krozel, J. A., Payton, D. W., and Tseng, D. Y. (1994) "Case-Based Computer Generated Forces," in *Proceedings of the Fourth Conference on Behavioral Representation and Computer Generated Forces.*

Mitchell, J., Payton, D.W., Keirsey, D.M. (1987) "Planning and Reasoning for Autonomous Vehicle Control," *International Journal of Intelligent Systems*, Vol. II.

Schwamb, K., Koss, F., and Keirsey, D. (1994) "Working with ModSAF: Interfaces for Programs and Users," in *Proceedings of the Fourth Conference on Behavioral Representation and Computer Generated Forces.*

7. Authors' Biographies

Michael D. Howard has been active in the study of Semi-Autonomous Forces and Autonomous Systems since 1988. Since the work described in this paper, he has become Principal Investigator of the Command Forces program, a new ARPA program. Mr. Howard holds the MSEE degree from University of Southern California (1986), a BSEE from Louisiana State University (1981), and a BA in English from St. Andrews College (1977). His research interests are in the areas of multi-agent coordination, autonomous systems, and adaptive planning.

Bruce Hoff is the Principal Investigator for the USMC-SAF project at Hughes Research Laboratories. At HRL Dr. Hoff has also worked on the ARPA autonomous vehicle project, and CGF Command Forces (CFOR). Dr. Hoff earned B.S. and M.S. degrees from Rensselaer Polytechnic Institute, and the Ph.D. degree from the University of Southern California, all in Computer Science. His research interests are in the areas of Artificial Intelligence, Autonomous Vehicles, Adaptive Control, and Computer Generated Forces.

David Y. Tseng is the Manager of Government Program Development for the Information Sciences Laboratory. Dr. Tseng holds a Ph.D. from the Polytechnic Institute of Brooklyn (1967), a S.M. from Harvard Univ. (1962) and a B.S.E.E. from the Univ. of Pennsylvania (1960). Dr. Tseng's current activities are in the areas of artificial intelligence, autonomous systems, synthetic environments, and information management. He is actively involved in developing programs that utilize artificial intelligence technologies to advance the capabilities of military systems.

Mobility Behavior in Dismounted Forces

George R. Mastroianni
US Army Natick RDEC
SATNC-YBH
Natick, MA 01760

Reed W. Hoyt
USARIEM
Altitude Phys and Med Div
Natick, MA 01760

Mark J. Buller
GEO-CENTERS, Inc.
190 N. Main St.
Natick, MA 01760

1. Abstract

We have been working to enhance the representation of dismounted soldier mobility behavior in the Integrated Unit Simulation System (IUSS). This paper reports the results of our analysis of the mobility behavior of twelve US Marines who walked 112 kilometers in Yosemite National Park over seven days in September 1994. Using GPS monitoring, we recorded the position of the Marines, the time, and their verbal description of the terrain as they completed this hike. Using the GPS records, trail logs, and topographic maps, we constructed a comprehensive description of the march rate and rest patterns of the Marines and of the trail grade, footing, and conditions. We analyzed the available data and established functional relationships among terrain conditions and the march rate and rest patterns of the Marines. We concluded that grade was an important determinant of march rate.

2. Introduction

The Integrated Unit Simulation System (IUSS) is a comprehensive small-unit simulation environment being developed by Natick Research, Development and Engineering Center and Simulation Technologies, Inc. (O'Keefe, 1994) IUSS includes sophisticated modeling of the physiological state of dismounted soldiers. Environmental conditions, mission demands, and terrain interact dynamically to alter the heat stress and energy requirements of dismounted soldiers in IUSS. The system tracks a variety of individual physiological parameters, such as core temperature, skin temperature,and heart rate, and computes appropriate changes in these parameters during simulation runs. This information is continuously available during the simulation and is used to alter the performance capabilities of simulated individuals.

2.1 Mobility Control in IUSS

Using the terrain databases provided to the system, IUSS computes the grade of route segments along which dismounted soldiers travel. (The segmentation of the route is determined by the magnitude of the grade change - the threshold for defining new segments is controllable). As simulated individuals travel the route the system uses the environmental conditions, terrain characteristics, and load to compute energy expenditure and heat production.

The system also uses the thermal transfer properties of clothing and equipment worn by the simulated soldier to calculate heat strain on the individual (Pandolf *et al*, 1986). When predetermined thresholds for core temperature are exceeded, the individuals performance is degraded.

2.2 Mobility Behavior of Soldiers

The existing computational approach to mobility control is adequate for making relative comparisons or trade-off analyses, but may not accurately represent the behavior of soldiers under operationally realistic conditions. The availability of more sophisticated information about how soldiers move across terrain would enhance both the analytic value of IUSS (or other simulation systems) and the realism of computer generated dismounted forces.

2.2.1 March Rate Control

Most studies of soldier locomotion have been conducted in laboratories using treadmills. March rate is typically controlled by the experimenter, leaving little opportunity for assessment of march rate under ecologically

valid conditions. The limited field data available suggest that soldiers may tend to maintain a more or less constant level of effort rather than a constant speed (Myles *et al*, 1979). In any case, it is apparent that march rate control is a complex and dynamic process affected by environmental conditions, individual physiological and psychological variables, mission requirements, and terrain characteristics. We used a field study to try to improve our understanding of this complex process.

2.3 Field Study

The study described below was designed to provide field data to further validate and refine the energy expenditure and heat production algorithms developed at USARIEM (US Army Research Institute of Environmental Medicine) that are used by IUSS. An additional effort was made to collect data on soldier mobility behavior during the study to begin the process of developing more complex soldier march rate control modules for IUSS.

3. Study Plan

The field study was conducted at Yosemite National Park in the fall of 1994. Twellve Marines traversed a pre-planned route covering approximately 110 kilometers while bearing loads of approximately fifty pounds. The groups were permitted to move at their own pace, but were required to arrive at checkpoints on particular days to accomplish resupply and data downloading tasks.

3.1 Data Collection

The main purpose of the effort was to study energy expenditure of soldiers and Marines engaged in a lengthy, rigorous hike at moderate altitude. Extensive measures of individual physiological fitness, including aerobic capacity and body composition, were made on each participant before the hike. Participants also ingested doubly-labelled water and provided periodic body fluid samples during the study to permit accurate measurement of energy expenditure. Dietary intake and heart rate were also monitored throughout the study.

Global Positioning System (GPS) fixes were recorded by the group frequently during the hike; an audio tape recorder was used to record trail descriptions and annotate rest break timing. The primary intent of gathering these data was to obtain high-resolution information to aid in the refining of our predictions of energy expenditure.

3.2 Analytic Approach

Since our sub-goal was to relate terrain characteristics to march rate, we had first to adopt a data processing strategy that would extract as much information from the available data as feasible. Since we had time-tagged GPS fixes, we used these fixes to develop our route profile. This route profile was the basis for most of our subsequent analysis.

3.2.1 Route Profile Construction

We began by plotting the route taken by the Marines on a topographic map. We then segmented the route into discrete pieces. The segments were chosen such that the time, elevation, and location of the beginning and end of the segment were known, and the segment contained terrain that was consistent in type and grade. Insofar as was possible with the data available, segments were chosen so that the average grade across the segment represented well most of the terrain in the segment. The topographic maps and the trail logs from the tape recorder provided the basis for these segmenting decisions. Because the terrain was quite variable, segments varied in length from a few hundred meters to several kilometers; the average segment length was just under one kilometer. Rest breaks also ended and began segments, even if they occurred in terrain that would not ordinarily have triggered a new segment. Figure 1 shows a frequency distribution of the segments we used in the analysis. This distribution illustrates that even though the terrain was very mountainous, most of the walking was done at low to moderate grades (0 - 10%).

3.2.2 Route Profile Analysis

Using the topographic maps, the trail distance from the beginning to the end of each segment was measured. The topographic maps were also

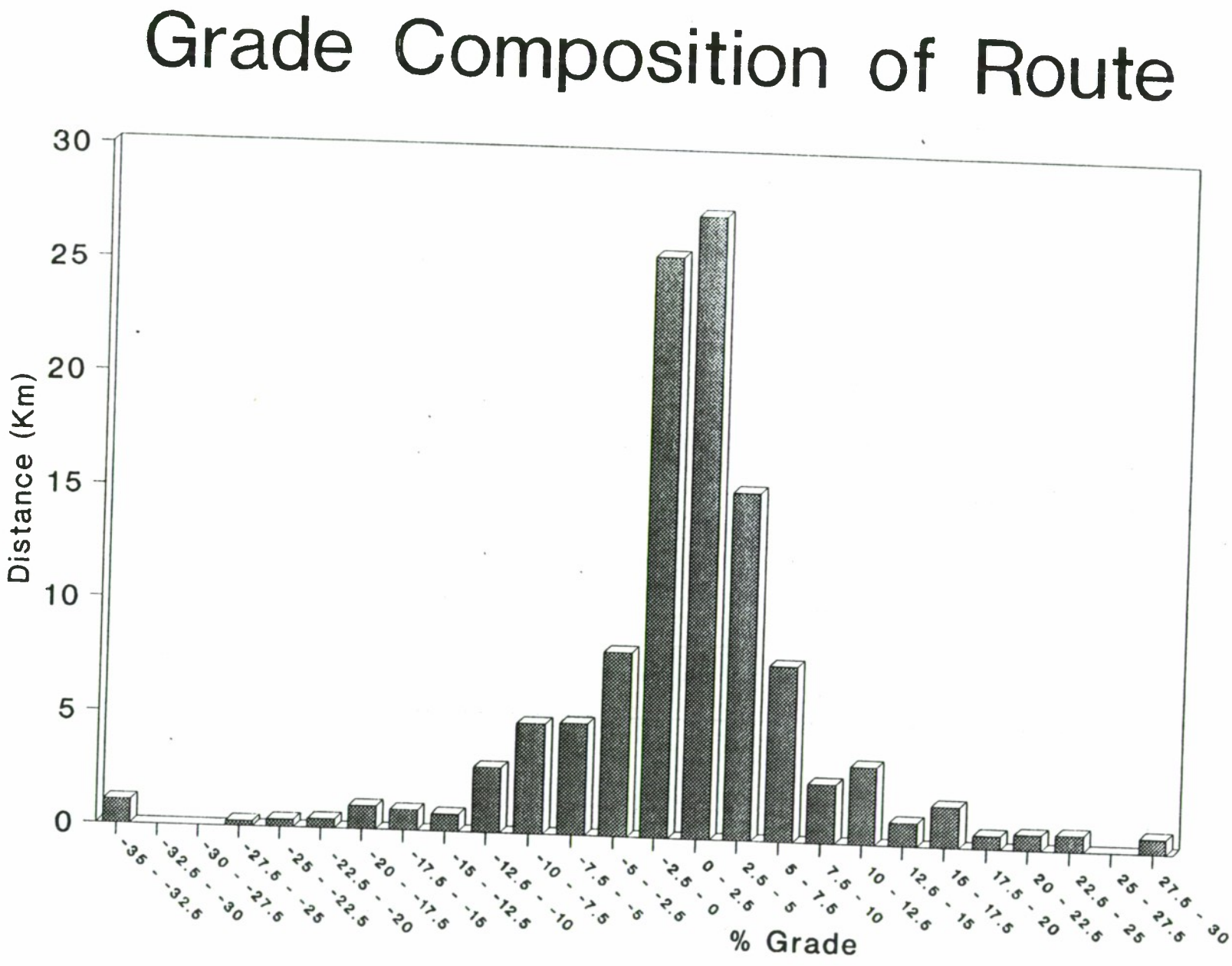

Grade Composition of Route
Distance (Km)
30
25
20
15
10
5
0
-35 - -32.5
-32.5 - -30
-30 - -27.5
-27.5 - -25
-25 - -22.5
-22.5 - -20
-20 - -17.5
-17.5 - -15
-15 - -12.5
-12.5 - -10
-10 - -7.5
-7.5 - -5
-5 - -2.5
-2.5 - 0
0 - 2.5
2.5 - 5
5 - 7.5
7.5 - 10
10 - 12.5
12.5 - 15
15 - 17.5
17.5 - 20
20 - 22.5
22.5 - 25
25 - 27.5
27.5 - 30
% Grade

Figure 1

used to validate the elevation estimates provided with the GPS fixes. (The contour interval of the maps was 40 feet.) When there was an elevation change over a segment, the distance used was the "slant distance" taking into account the elevation change. Velocities for each segment were then computed by dividing the distance traversed over a segment by the time taken to move from the beginning to the end of the segment. Grade was computed by dividing the elevation change from beginning to end of the segment by the total length of the segment.

4. Study Results

Our analysis has thus far focussed on two issues: the relationship between grade and velocity and the timing of rest breaks.

4.1 Grade and Velocity

How fast an individual walks is, as we mentioned above, determined by a complex constellation of variables. Because grade is closely related to the work output required, which is in turn closely related to heat production and fatigue, grade is likely to account for a fair amount of the variation observed in velocity. Moreover, grade can be quantified relatively easily, unlike the less tangible variables such as motivation or perceived mission demands.

4.1.1 Steepness and Velocity

Figure 2 shows a scatter plot of all the segments defined along the route. This plot contains segments from seven different days. The intensity of effort varied dramatically across these different days of the hike; some days were characterized by a relatively leisurely pace and other days (such as the day the group lost the trail and struggled to reach a camp before nightfall) were characterized by exhausting effort. Median velocity for the uphill segments was 0.93 m/s; for downhill segments, the median was 1.15 m/s. The vertical scatter of the points on Figure 2 for a particular grade shows that while grade clearly accounts for some of the variability in velocity, there are also other factors at work.

Figure 3 shows a plot of average velocity as a function of grade using bins of 2% . (Due to the low density of data at grades higher than 12%, individual points are plotted beyond these values). This plot shows nicely the effect of increasing uphill grade on velocity. The decrease in velocity with increasing grade is less pronounced for downhill grades, probably reflecting the more variable effects of footing and stability on velocity when moving downhill, instead of the work output limitation on velocity on uphill segments.

4.2 Rest Break Patterns

In laboratory studies of locomotion on treadmills, a work-rest cycle of 50 minutes walking and 10 minutes rest is often assumed. . Our Marines took thirty-seven breaks over the seven days of hiking. These breaks included brief pauses to adjust equipment, lunch breaks, and rest breaks. Distances traversed before taking a break ranged from a few hundred meters to more than six kilometers; times ranged from a few minutes to nearly ninety minutes. Figure 4 is a scatter plot of the breaks taken by our Marines.

As with the relationship between grade and velocity, the factors that determine when a group of soldiers will pause for a break are apparently quite complex. There is a tendency for higher grades to be associated with a shorter time between breaks and a shorter distance covered between breaks. Figure 5 shows the average time and distance since last break for intervals with average grades above 5%, less than -5%, or between -5% and 5%. Uphill grades are associated with a decline in performance relative to flatter terrain, and downhill grades show a similar but smaller change. This effect is greater for the distance covered than for the time taken before a break, suggesting that the group sacrificed some velocity before taking a break under stressful conditions.

5. Discussion

The results of this study have shown us that it is possible to acquire useful and interesting information about dismounted soldier mobility behavior in the field using relatively unobtrusive and inexpensive methods. The continuing improvement in the quality and availability of

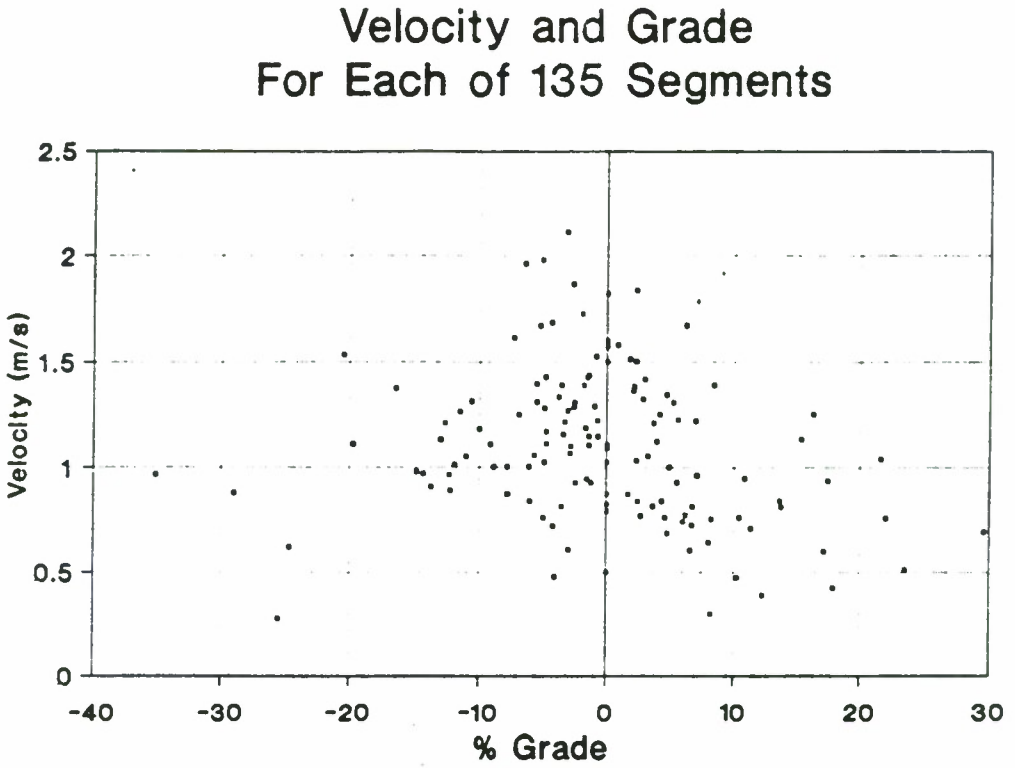


Figure 2

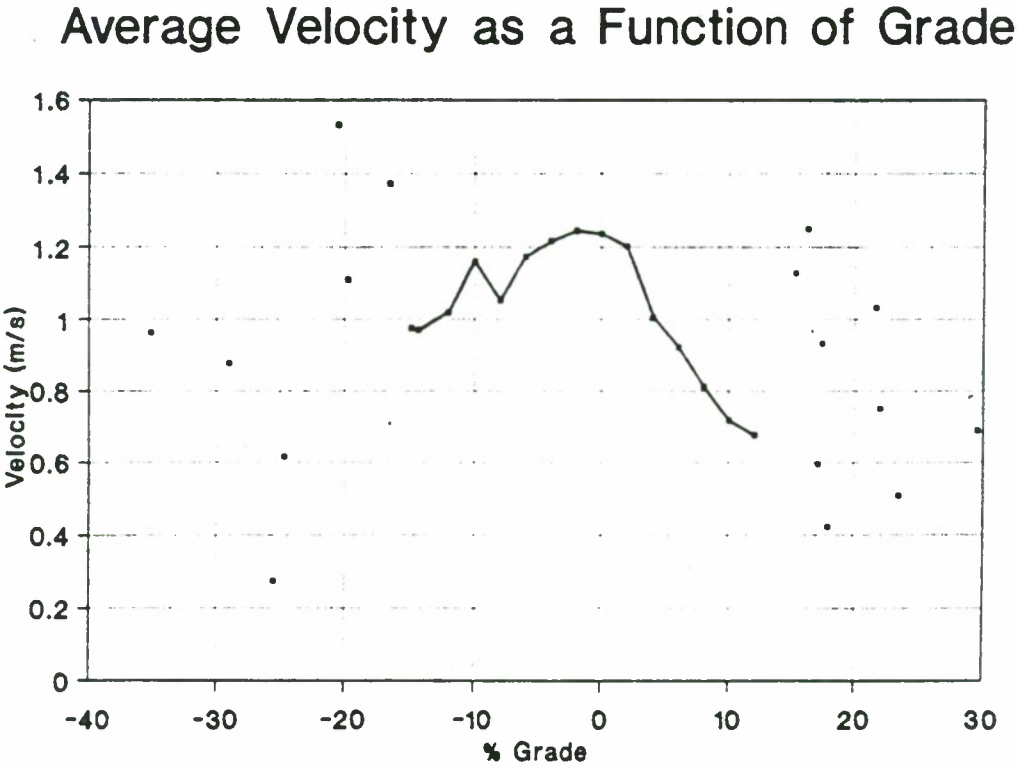


Figure 3

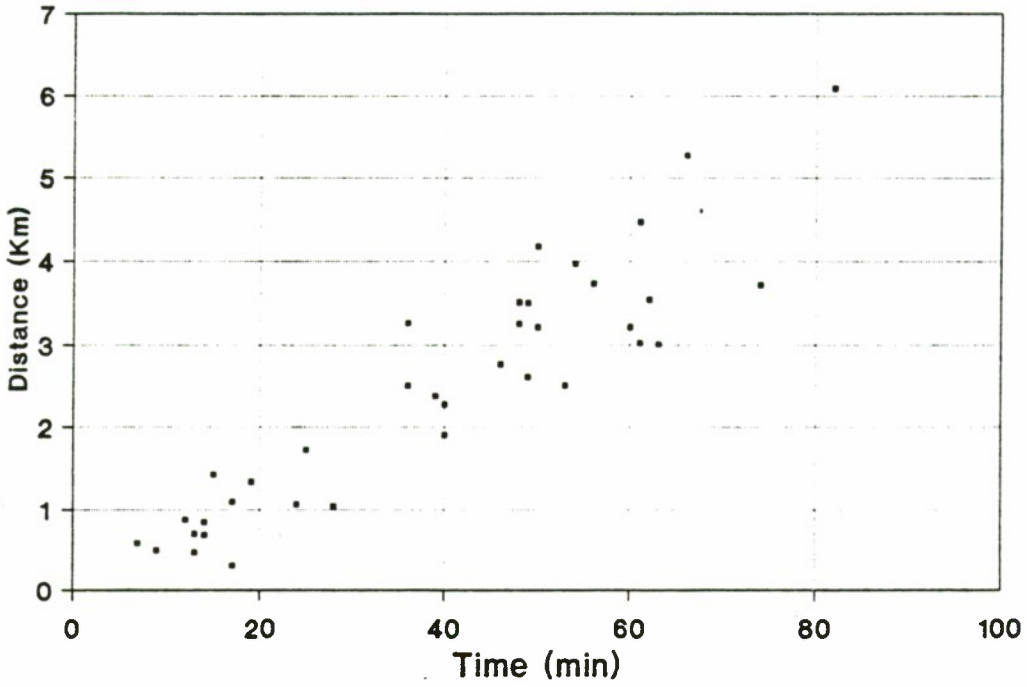


Figure 4

This figure shows the distance traveled and time elapsed between breaks. All breaks taken over seven days are included.

Distance and Time Since Last Break (SLB) vs %Grade

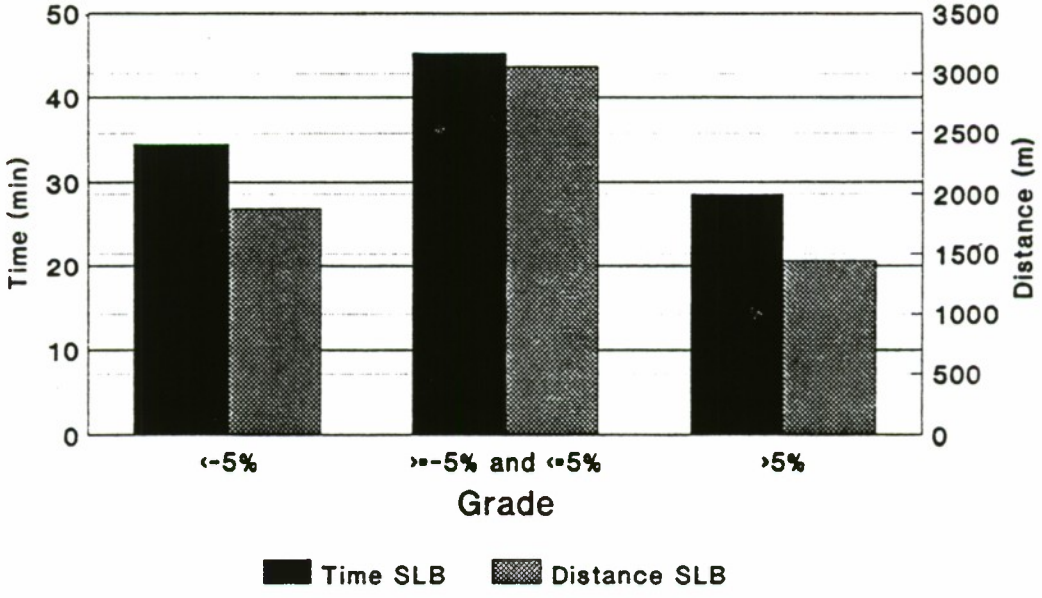


Figure 5

GPS equipment should make studies like this one even easier in the future.

5.1 Study Limitations

The generalizability of the results of this study is limited in several respects. The Marines who took part in the study were young, fit, well-trained and led by experienced mountaineering instructors from the USMC Mountain Warfare Training Center. The mission scenario did not include a tactical context; the guidance given the mission planners was to attempt to plan a schedule that would ensure a variety of work intensities over the course of the hike. The loads carried by the Marines were moderate but significant. The information extracted from this study clearly cannot be considered to be typical of movement rates under very different conditions in the absence of further empirical investigation.

Future studies undertaken to extend the findings of this study should include variations in the tactical context of the study, the group size and group composition, and mission duration. The tactical context is critical because it introduces many more constraints on movement rate than existed in this exercise. Our Marines were not attempting to move stealthily, nor were they attempting to scan the terrain for obstacles or enemy. Tactical operations also demand pauses for communication with other units, as well as a variety of non-locomotion tasks that may sap strength and alter movement rates relative to our sample.

The group size and composition clearly affect performance on a long hike. Group composition is important insofar as age, fitness, and experience determine the physical resources available to the group. The leader of the group sets the pace but does so with due consideration of the capabilities of the members of the group and the potential for sustaining the pace long enough to complete the mission.

The duration of the mission is a critical factor in determining movement rate. Wise leaders husband resources effectively to ensure the mission can be accomplished. Both pace and break timing would have been very different if the mission had been much longer or shorter than seven days. Taking into account the cumulative effect of stress and exertion when modeling soldier mobility performance is an important area for further work.

5.2 Future Plans

We intend to continue our analysis of the data described here by systematically attempting to explain some of the variation in movement rate not explainable by grade. Footing or soil composition and the cumulative effects of fatigue over the course of the seven days are possible avenues of investigation.

We plan to integrate the extensive physiological data collected during the study into our mobility analysis. Heart rate recording, for example, provides us with a way to relate effort to pace and break timing in a much more rigorous way than is possible using only the terrain characteristics.

We also plan to extend our analysis to the group of 12 Army Special Forces soldiers that negotiated the route independently. This will afford us a unique opportunity to compare the performance of two different and independent groups on exactly the same terrain, under the same environmental conditions. An assessment of the magnitude of the differences in mobility performance between the two groups will provide an initial check on the generalizability of our findings.

6. Acknowledgements

The authors would like to acknowledge the assistance of John O'Keefe and John Ward of Natick RDEC; Victor Middleton of Simulation Technologies, Inc.; the staff of the USMC Mountain Warfare Training Center; and the Marines and soldiers who performed with good cheer and incomparable professionalism.

7. References

Myles, W.S., Eclache, J.P., and Beaury, J. (1979) Self Pacing During Sustained, Repetitive Exercise. *Aviation Space and Environmental Medicine*, 921-924, September 1979.

Pandolf, K.B., Stroschein, L.A., Drolet, L.L., Gonzalez, R.R., and Sawka, M.N.

(1986) Prediction Modeling of Phyisiological Responses and Human Performance in the Heat. *Comput. Biol. Med.* Vol 16, No. 5, 319-329.

O'Keefe, J.A. When Are Data Base Simulations Not Good Enough? (1994) Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, May 4-6, 1994.

8. Authors' Biographies

George R. Mastroianni, Ph.D. is a Research Psychologist in the Human Factors and Ergonomics Branch, Behavioral Science Division, Science and Technology Directorate, Natick Research Development and Engineering Center. His research interests are in the area of human performance measurement.

Reed W. Hoyt, Ph.D. is a Research Physiologist in the Altitude Physiology and Medicine Division, Environmental Physiology and Medicine Directorate, US Army Research Institute of Environmental Medicine. His research interests are in altitude, metabolism, and human performance.

Mark J. Buller is a Human Factors Engineer at GEO-CENTERS, Inc., working under contract to Natick RDEC. His research interests are in artificial intelligence and human performance.

A Behavioral Approach to Fidelity Requirements for Simulation of Dismounted Combatants

Robert T. McIntyre, III
Simulation Technologies, Inc.
1604 Cedar Lane
Raleigh, NC 27614-9350
rmcintyr@snap.org

Victor E. Middleton
Simulation Technologies, Inc.
111 West First Street, Suite 748
Dayton, Ohio 45402-1106
middletv@rcinet.com

1.0 Abstract

The introduction of the individual dismounted combatant into distributed simulation has generated considerable controversy with respect to questions of simulation fidelity and resolution. Model resolution is a function of several factors, among them technology constraints driven by hardware and software capabilities, the availability of data to support specific simulations, and the compatibility requirements of different simulation modules.

While these factors are important, an often overlooked concern is the ability of simulated entities, such as dismounted combatants, to provide the necessary behavioral cues to other participants in a distributed simulation. This paper examines an approach to building simulation scenarios based on decision trees that represent entity options in reaction to the observed behaviors of others. These decision trees are associated with specific battlefield operation systems tasks of the Army Training and Evaluation Program. The level of detail of entity action required to trigger these decision trees provides one measure of the resolution required for associated simulations.

2.0 Introduction

The growth of Distributed Interactive Simulation (DIS) has injected new fervor in an old debate: how do we define the adequacy of model fidelity and resolution. As little as 10 years ago, many were content to answer this question by saying: Whatever I can get! We were driven more by software and hardware constraints than by questions of relevancy -- what was possible as opposed to what was needed to get the job done. While hardware and software constraints haven't gone away, current and near-term capabilities have pushed the realm of the possible enormously. While in some ways this makes our job easier, it does present more choices. When the menu had only a couple of items on it, picking one wasn't hard.

Today we are faced with the need to make multiple, informed choices concerning the degree of resolution we employ. What is the basis for these choices? The DIS world has been accused of opting for show, picking that level of resolution that grabs the attention of the observer. This is justified if a distributed simulation exercise is successful only if it can capture the participants, interjecting them into the simulated world. A primary goal of DIS is the seamless transfer of behaviors from the simulated battlefield to the actual. The difficulty is that we can get caught up in the race to make our exercises and demonstrations increasingly spectacular, and find our efforts driven by the ever-more impressive capabilities of display technologies as opposed to the substance of our underlying simulations.

The thesis of this paper is that, ultimately, issues of resolution and fidelity must be decided concerning the level of detail needed to simulate the behavior, and therefore the performance, of the simulation entities. The network architecture that allows these entities to interact must incorporate representations of the battlefield environment to whatever level of detail is sufficient to determine the dynamic response of each entity to that environment, and, to each other.

3.0 Background

It is important to say that the authors of this paper take a rather parochial view of their subject matter. We are concerned with modeling and simulation (M&S) of the individual dismounted combatant, and in particular, the application of M&S tools to solve the problems of the Soldier as a System. This alone provides a first, gross filter concerning questions of model resolution: to whit, models that don't represent the individual versus those that do. The former are outside the scope of this paper, which is directed to addressing issues involved in simulating the Soldier System, and specifically the individual dismounted combatant. The authors have been heavily involved in the development of the

Integrated Unit Simulation System (IUSS) for the US Army Natick Research, Development and Engineering Center (Natick). The IUSS provides an architecture for the integration of individual models simulating different aspects of battlefield environment, and thus represents in microcosm the problems DIS has in the coordination of large-scale distributed operation of multiple simulations. The IUSS also provides a context for this paper's discussion of resolution issues, if only because development of the IUSS has forced the authors to confront these issues head-on.

3.1 The Soldier System

Historically, equipment for the soldier has been developed through separate, distinct initiatives. M&S tools to support such development and to represent the result in distributed simulations, have traditionally followed this division, with separate models for ballistic weapons, individual protection, etc. While the single piece or "eaches" development may have been carefully planned and implemented concerning their individual goals, the result was still an overwhelming collection of disparate items, contributing to a grossly overloaded soldier. The current recognition of the need to treat the soldier as a "Soldier as a System" comes from the realization that the soldier, his weapons, protective gear, and other supplemental equipment must function together as a integrated system, and hence must be designed, evaluated, and maintained as a system. This systems approach to the soldier, and a corresponding obligation to look comprehensively at the entire battlefield environment, drive a need for consistent resolution across multiple facets of battlefield operations.

Estimation of any single aspect of the battlefield environment (e.g., combat systems, personnel attrition, performance degradation, thermal stress) does not provide a comprehensive understanding of a unit's (or an individual's) ability to perform its mission. For this reason, an integrated approach to simulation of the battlefield environment is required, combining the effects from a variety of factors, while permitting assessment of the contribution from each. Such an approach should facilitate the incremental inclusion of additional factors, allowing the construction of increasingly more complex scenarios (i.e., scenarios with a higher degree of resolution). These factors can be represented by integrating features of classical models, each of which emphasis a single specific aspect of the battlefield (e.g., combat systems, performance degradation, thermal stress), into a cohesive architecture to provide a comprehensive understanding of an individual's ability to perform combat mission tasks.

This approach conforms to DIS philosophy, permitting disparate models or simulations to interact with each other. This approach must, however, also ensure that such disparate players share some level of common assumptions. Furthermore, if these models do not operate at common levels of fidelity or resolution, some mechanism must be provided to aggregate or de-aggregate their output to a common interface.

3.2 The Integrated Unit Simulation System

Natick, supported by Simulation Technologies, Inc. (and, in particular, the authors of this paper), has developed the Integrated Unit Simulation System (IUSS), to provide a comprehensive analysis environment for the evaluation of Soldier System's survivability and effectiveness. The IUSS design has paralleled the evolution of the Soldier System concept, combining historically disparate models of different aspects of the soldier and the soldier's combat systems into an integrated representation of the battlefield. The IUSS provides an open, extensible architecture for the unified representation of all aspects of the modern battlefield: threats, personnel, equipment, and environmental factors. The IUSS is designed to accommodate bundled access to current and evolving methodologies, providing a flexible simulation package. In particular, this design facilitates the integration of disparate models and has had to contend with reconciling disparate levels of model resolution, either by filtering or averaging fine resolution data to fit more coarse input requirements, or augmenting and extrapolating coarse data to fill up fine grid requirements.

The common basis for all IUSS effects is the psycho-physiological state of the individual soldier, and how that state relates to the soldier's task and mission performance. Unit performance depends on the performance of the unit's components (the individual combatants), just as mission success depends on the successful completion of mission components (the mission tasks). The IUSS represents missions as task networks, with the tasks following the form of the Battlefield Operating Systems/Tasks (BOS-T) of the Army Training and Evaluation Program (ARTEP) manuals. The ARTEPs describe practically any task a soldier may be

called upon to perform, subject to the dictates of mission, enemy, terrain, troops, and time available (METT-T). Using the BOS-T/METT-T paradigm allows the IUSS to dynamically adjust task performance as appropriate based on the behaviors of the individuals performing specific tasks throughout the execution of a simulation scenario. Developing the mechanisms to control this dynamic adjustment has led the authors to conclude that the detail needed to represent the behaviors of individuals, and the environmental cues that drive those behaviors, is the most important consideration in determining appropriate model resolution.

3.3 Vocabulary

Fidelity and resolution have different definitions to different individuals and therefore are ambiguous terms. To avoid being side-tracked by a debate that is not particularly germane to our purpose, we propose to avoid defining them explicitly. Instead, we focus on making a differentiation between two distinct types of resolution (or fidelity). The first of these types is resolution of kind, and the second is resolution of degree. Resolution of kind refers to the kinds of things we are modeling, the types of detail we wish to include. Perhaps the best known example of resolution of kind is classification of models by echelons of combat represented. Under this scheme, theater or global campaign models represent the lowest degree of resolution, while models of squad or individual operations the highest. Resolution of degree refers to quantitative measurement accuracy: spatial measurements in terms of millimeters represent higher resolution than measurements in terms of furlongs (or kilometers), temporal measurements in picoseconds higher resolution than those in years.

4.0 Quantification Issues

Clearly, there is a loose correlation between the two types of resolution described above. When modeling theater echelons of combat, we seldom concern ourselves with terrain at the millimeter level or time at the pico-second level. We tend to measure macro-level phenomena with coarse measures and micro-level phenomena with finer ones. Still, we occasionally fall prey to the desire to have a really impressive display, and show the fine level of detail where it may not be required, or worse, where it may not be consistent with other linked models -- the process sometimes characterized as: measure with a micrometer, mark with a crayon, cut with a chain saw. This leads us to a first principle in choosing an appropriate resolution: consistency. One should not attempt to suggest a spurious accuracy or validity for simulation outcomes by providing an output format or display which is not consistent with the resolution of the underlying processes -- chain saw cuts require drawing your output display with a thick, blunt crayon.

4.1 Simulation Without the Man-in-the-Loop: Decision Trees

As mentioned above, much of the concern over model resolution in the DIS world has to do with the question of the show, the visualization of the simulated environment that is presented to the DIS participant, and as noted, this can be important to the success of a DIS exercise. It is not, however, important to the discussion in this paper. The IUSS, unlike many of the components of the DIS environment, does not provide for a man in the loop during actual simulation execution. This restricts the scope for issues of model resolution and fidelity to the requirements of the simulation, as opposed to the perceptual needs of a human observer. This is not to say that the IUSS does not consider human sensory capabilities. Rather, the IUSS restricts the consideration of these capabilities to the level of detail specific object procedures require. For example, to determine if a simulated soldier sees an enemy target, a number of different algorithms can be called into to play. The simplest determines if a line of sight exists between the soldier and the target, while there are also complex calculations that consider the contrast sensitivity between the target and its background, the possible confounding presence of obscurants, and the ability to augment human capability with a variety of optical sensors. Ultimately, whatever algorithms are used, the simulated process of seeing will come down to a yes or no decision concerning any particular target.

In fact, all the behaviors of the simulated soldiers ultimately reduce to a set of such decisions: describe environmental features and/or entities by classification by deciding the status of specific state variables (e.g., -- is target present, does line of sight exist to target, is there sufficient ambient light to observe target, is the target obscured by smoke or foliage, is target signature sufficient) and deciding what the appropriate entity response is to that classification. Such decision processes can be arranged in a hierarchy or decision tree,

analogous to increasingly finer filters to sift out behavioral choices. These decision trees provide a natural classification scheme for levels of resolution, corresponding to the levels of the decision tree itself. Furthermore, this supports the concept of variable resolution requirements, since a simulation need provide only enough information to support whatever levels of decisions are deemed appropriate to a specific application. The level of decision depends on the types of behavior being simulated, and there is no need to assume that behavior must always be studied concerning individuals -- unit behavior may be quite adequate for many applications.

4.1 An Example: Resolution in Kind

The IUSS provides the ability to model multiple echelons of combat, but as stated above, the focus of the authors and the primary design consideration of the IUSS is explicit representation of the individual, operating in the context of platoon or squad sized mission tasks. These tasks take place in a battlefield environment, specifying the factors to be played (resolution in kind), calling for selecting aspects of the battlefield and the soldier's interaction with it, including the environment (terrain, weather, time of day), the threat, the mission, and the composition of the forces assigned that mission. To over-simplify, those factors that describe the environment are cues to the soldier's behaviors, the behaviors themselves are specified by the mission task structure, and within that structure, further refinement of individual procedures or options.

IUSS input scenario mission descriptions need to provide detail down to the level of individual tasking, accompanied by the first level decision trees that specify individual behaviors: the rules of engagement and the specific tactics to be employed for each mission task or phase. Since the IUSS is a two-sided simulation this level of detail needs to be given for both Blue and Red forces to the extent to which either or both are to be played. There is, of course, no requirement to include the same level of forces if there is no need to play explicit force-on-force interaction, i.e., Blue forces can be played without any Red opposition, or opposed only by indirect fire, or opposed by Red Forces represented at the same level as the Blue, according to the objectives of any given scenario.

The underlying basis of the IUSS is the psycho-physiological state of the individual. Following the object oriented paradigm, this state is represented by a number of different attributes belonging to the soldier object. These attributes include such things as a thermal regulation, ballistic injury mechanisms and chemical intoxication accumulation. Each of these attributes can be ignored or turned off to speed model execution times as appropriate to specific simulation objectives. The factors above may each be broken down into sub-factors; the choice of which factors and sub-factors to include in scenario inputs is driven by the needs of the attribute update processes.

4.3 An Example: Resolution in Degree

One of the surest ways to provoke spirited debate among players in the DIS world is to raise the topic of terrain resolution. How fine a terrain grid is needed for a valid battlefield for the individual dismounted combatant? A soldier in combat can and will make use of very meager cover if that is what's available. To represent that soldier faithfully, do we have to include any rock or tree that he might hide behind? Or the height of the individual blades of grass through which he might crawl? There is no doubt that a soldier being shot at may become very familiar with his terrain on a very micro-level.

To make the debate even more spirited, introduce the question of the simulation application. One contention holds that if the application is training, we need to very faithfully, and in great detail (1m.<), replicate the terrain to present a valid virtual environment for soldier training. This theory may be true, but experience suggests otherwise, at least as a blanket assumption. The function of training is to induce transfer of behaviors from the training environment to the actual environment; the cues that are presented to the trainee need only be adequate to facilitate that transference. As an example, the use of silhouette targets appears adequate to train riflemen, the advantages of being able to present them a more detailed articulated body picture have not been demonstrated for that purpose. On the other hand such an articulated body picture may be required to support training in another arena, for example, the use of hand signals. Is fidelity enhanced by using digitized terrain from actual locations? And if so, how fine a grid must be used. Experimentation with such data is extremely useful with respect to trying to verify the correctness of many of our models that use terrain data, does its use increase the predictive validity of those models? To what extent can we get by with the use of geo-typical terrain, or

random draws from statistical distributions of terrain features? If we use statistical distributions, how many sampling points should we use for our grid? It is hard, if not impossible to answer such questions in the abstract.

However, for a specific simulation experiment we can fall back on the concept of behavioral requirements. If (as the author's are currently trying to do) we are investigating the self-pacing behavior of marching soldiers as a function of their load, the ambient weather, terrain grade and surface type, we can determine from actual field trials the resolution required. We can observe the changes in these terrain factors that are sufficient to cause the soldiers to alter their behavior, either by changing their rate of speed, the frequency with which they take breaks, or their choice of route.

A fundamental question we need to ask regarding increasing resolution of terrain, and any other factor, is: if more detail is provided in the simulation, do I know how the individual will alter his or her behavior in response to that increased detail? We can populate the forest with trees and rocks to incredible levels of detail -- do we know which ones the soldier will pick for protection? Or for how long?

5.0 Summary

The admittedly hard questions as to what is adequate resolution in our models and simulations are at least approachable if we consider them from a behavioral perspective. Simulation is, after all, a process for attempting to assess how entities interact with their environment. This interaction, in the case of the dismounted combatant, is equivalent to soldier behaviors. Assessment of model resolution requirements should be driven by what is required to trigger or alter those behaviors. It is enormously easier to include high resolution terrain data bases in our models than it is to include high resolution behavioral data bases. What is the benefit of the one without the other?

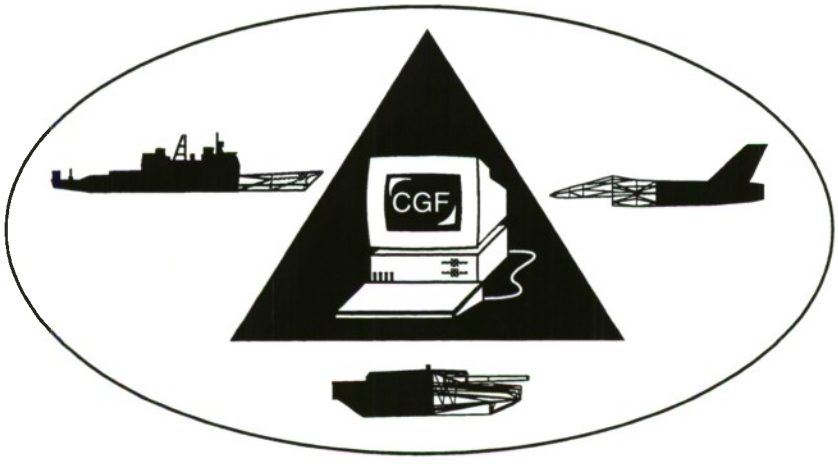
6.0 Acknowledgments

This work was supported primarily under contract DAAK60-94-C-1061 with the U.S. Army Natick Research Development and Engineering Center. The authors would like to acknowledge the encouragement of John O'Keefe, U.S. Army Natick RDEC, Daniel E. Mullally, Jr., Institute for Simulation and Training, University of Central Florida

7.0 Authors Biography

Robert T. McIntyre, III, is the Program Manager for the Integrated Unit Simulation System and is the Vice-President for Operations for Simulation Technologies, Inc.

Victor E. Middleton, is a Senior Operations Research Analyst under contract to Simulation Technologies, Inc. Mr. Middleton earned an MS degree in Mathematics from Michigan Technical University in 1975 and an MS in Applied Mathematics from Michigan State University in 1978.


CGF

Simulation of Suppression for the Dismounted Combatant

Victor E. Middleton
Simulation Technologies, Inc.
111 West First St.
Dayton Ohio 45402
middletv@rcinet.com

W. M. Christenson
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311-1772
wchriste@cs.ida.org

John D'Errico
Dismounted Battlespace Battle Lab
Fort Benning , GA 31905
derrico@benning-emh1.army.mil

1. Abstract

Recent advances in the sophistication of distributed interactive simulation (DIS), coupled with the development of the concept of the Soldier as a System, have produced a requirement for greater fidelity in soldier simulation. Historically, the individual dismounted combatant has not been a key player in distributed simulations or analytical studies. When individuals were represented in such simulations, they were generally treated simply as smaller, slower, less lethal versions of other simulation entities, e.g., an unarmored tank. Such a representation does not allow the examination of the dynamics of small unit conflict required for training, or for research and development in support of the individual soldier.

To maintain even face validity, simulations of individual combatants must characterize the individuals' interaction with the environment and with other individuals. This requires explicit representation of the soldier's dynamic response to the battlefield, and the effect of the soldier's behaviors on the outcome of the conflict. In particular the role of suppression is a significant component of this interaction, but one which as yet we have been to capture effectively in M&S. The authors' approach to simulation of suppression begins with a definition the problem, identifies critical factors, and suggests priorities determined by current analysis.

2. Introduction

The current emphasis on the role of the dismounted combatant in distributed interactive simulation (DIS) is part of a revolutionary view of the individual soldier: the concept of the soldier as an integrated weapon system. This revolution is itself outside the scope of this paper, (interested readers are directed for example to: Haley et. al., 1992, or US Army Materiel Command, 1992), but its consequences have forced the modeling and simulation (M&S) community to deal with the individual soldier in far greater detail than we ever have before.

To maintain even face validity, simulations of individual combatants must clearly characterize the individuals' interaction with the environment and with other individuals. Suppression is a significant component of this interaction, and one which as yet we have been either unable or unwilling to capture effectively in M&S. While the authors are not so bold as to suggest that we have a solution to this problem, we would like to suggest an approach to simulation of suppression with respect to the dismounted combatant. We will begin with a definition the problem, identify critical factors, and present a plan of attack with priorities determined by current analysis initiatives.

2.1 The Authors' Perspective

Our perspective on the problem is, of course, biased by our background. The authors have all been heavily involved in the development of the concept of the Soldier System and the application of that concept to the Soldier Integrated Protective Ensemble (SIPE) Advanced Technology Demonstration (ATD) and the Twenty First Century Land Warrior (21 CLW) Integrated Technology Program. The need for analytical support for Soldier System initiatives has motivated a good deal of model development; of special interest to the authors are the US Army Natick Research, Development, and Engineering Center's Integrated Unit Simulation System (IUSS) (as described in: Middleton, 1992; Middleton and O'Keefe, 1993) and other high resolution combat simulations used in support of the US Army Infantry School's Dismounted Battle Space Battle Lab at Ft. Benning Georgia.

Our approach develops a simulation of suppression within the specific context of the

IUSS, although the methodology proposed is not unique to that context.

2.2 The Paper's Purpose

The purpose of this paper is to initiate a dialogue within the M&S community by explaining our approach and soliciting the opinions of others. Interested readers are encouraged to contact the authors with comments, opinions, additions, deletions, sources of data and/or methodology, and suggestions as to possible alternatives to that approach. After developing a framework for the representation of suppression, we hope to achieve general consensus with the community on that framework, develop and submit strawman values as appropriate to the methodologies decided upon, and iterate a process of analytical application and review of the results to evolve the sophistication of our representation while continuing the community's consensus.

3. Statement of the Problem

The concept of suppression is an integral part of combat at the level of the individual dismounted soldier. The ability to interrupt, impede, or suspend enemy operations may satisfy mission objectives as well (or better than) the destruction of enemy forces. Certainly in past wars suppressive fire played a role that was at least as significant as that of accurate aimed fire. However, small arms R&D studies and analyses have not reflected this fact. The vast majority of such studies have been directed towards the improvement of weapon accuracy, with relatively short shrift given to the investigation of suppression.

Why is this so? First, improving the accuracy of weapons fire is an undeniable benefit (or at least an unassailable goal). Secondly, modeling and simulation of weapons accuracy is something we (the M&S community) know how to do. Our models are based on well understood concepts of physics and the behavior of projectiles; we are comfortable in the acceptance of the accuracy of our results.

By contrast, we cannot (or at least haven't yet) even agree on a suitable definition of what suppression is, much less arrive at acceptable models of how it is achieved and how to represent it in simulations of combat.

3.1 The Challenge

We can no longer afford the luxury of ignoring these issues. We are faced with an urgent requirement for high fidelity simulation of the individual soldier, and that requirement cannot be met without explicit consideration of the role of suppression in determining the outcome of conflict. This requirement is driven by the downsizing of our military and the evolving nature of military mission, e.g., the need to examine smaller scale conflicts and operations other than war. The urgency is fueled by a need for more efficient application of resources for R&D, training, and operational support. Fortunately, this urgency is accompanied by enabling technologies: recent advances in the sophistication of distributed interactive simulation (DIS) and other hardware/software tools

The challenge we face now is how to best apply these technologies in the present era of shrinking budgets and expanding expectations. We need to achieve cost-effective improvements in our ability to simulate the individual dismounted combatant and, by extension, the representation of that combatant in DIS. The lack of unlimited resources means that we cannot achieve the additional fidelity we require by simply adding detail to our simulations. We must choose carefully what we add, and just as carefully what we leave out. Clearly, the authors (having undertaken this task and not being terribly enamored of unnecessary work) believe that suppression is one of the things we must add. That being the case, it now becomes our duty to explain exactly what we mean by simulation of suppression, to describe our rationale and objectives, and to provide our approach for achieving those objectives.

3.2 Simulation Rationale and Objectives

First, why are we tackling this problem? As stated above, suppression and the role of suppressive fire are important parts of small unit engagement, but two specific M&S initiatives, both involving application of the IUSS, are behind our current efforts.

The first of these is development of a computer generated force (CGF) module for the individual combatant. CGF forces who don't react "realistically" when being fired upon by other DIS entities, won't pass the initial test of surface believability. Our first goal in modeling suppression, therefore, is to ensure that CGF dismounted combatants provide "realistic" behavioral cues for other DIS entities. At a minimum we want them to react to immediate threats by appropriate avoidance behaviors, e.g., to duck, to move more cautiously and with effective use of cover.

The second M&S initiative of immediate concern is analytical support to the 21st Century Land Warrior (21 CLW) Integrated Technology Program. The 21 CLW program, and the associated Generation II Soldier Advanced Technology Demonstration (GEN II ATD) are the current instantiation of the Soldier System concept. These programs seek to examine the benefits in over-all soldier capability achievable by integrated application of technologies available now and in the near-term. This includes development and implementation of new weapon systems, sensors, communications, individual protection, and associated training and doctrine.

These programs acknowledge that Soldier System lethality, survivability, mobility, command and control, and sustainability are all intertwined, and that integrated improvement of these capabilities will produce synergistic benefits; the whole is greater than the sum of its parts. As an example, making the soldier more lethal can increase the ability to suppress enemy fire, making the soldier more survivable, which could lead to decreased weight required for individual protection, which could increase soldier mobility, which could allow faster closure with the enemy, leading to the ability to concentrate more fire on the enemy while reducing exposure to the soldier, which in turn increases lethality and survivability, etc.

Analysis in support of 21 CLW provides us with another set of objectives (albeit ones which overlap with CGF requirements) for simulation of suppression. To establish the benefits from proposed improvements in the Soldier System, we must measure the performance of these improvements and compare them to some baseline, i.e., determine deltas between the current (95 soldier) and the 21st Century Land Warrior capabilities. This requires determining the relationships between Soldier System MOE's (e.g., mission completion times, percent mission tasks completed, casualties, ammunition expenditure rates, soldier physiological state variables) and soldier behaviors as constrained by battlefield factors and as affected by the soldier's ability to employ technology.

This paper examines a set of key battlefield factors which contribute to combat stress or other measures of combat intensity, and suggests that these factors and their effects be loosely defined as suppression. In this context, our goal in outlining a framework for representation of suppression is accomplished by defining that set of factors, estimating their effects on soldier behaviors, and thus their effects on the analytical MOE's associated with those behaviors. Within this framework we then have the ability to examine how 21 CLW technologies can mitigate suppression as it affects friendly forces, and induce or exacerbate it with respect to enemy forces.

4. Approach

We begin by discussing what we mean by the term "suppression", at least within the confines of this paper. We define "degrees" of suppression in terms of the effects of such suppression on Soldier System performance and combat outcomes. We establish a set of battlefield factors or combat stressors which may induce suppression, and conclude by suggesting how our simulation methodology can represent the translation from the causes of suppression to observable and quantifiable effects.

4.1 What Is Suppression?

Our original discussions with combat veterans began with a very narrow definition of suppression: it is that instinctive reaction to an environmental cue, such as a nearby bullet impact, which causes the soldier to immediately seek the lowest possible profile, stop all movement, and in general try to disappear into the ground. Although this behavior basically implies complete disruption of combat operations while it lasts, this period is generally of very short duration. A more difficult question

is whether the residual effects of the original cue linger as some degree of combat stress, based at least in part on the intensity of conflict and the presence or absence of other combat stressors.

If we were to adhere to a narrow definition of suppression, concerned only with that initial brief reaction, our job would be relatively easy, modeling suppression as a brief cessation of activity when a "near" miss occurs. This narrow definition still leaves room for argument: exactly what is a "near" miss (i.e., which environmental cues will trigger the reaction), exactly how long "brief" is (i.e., how long do suppressive effects last). Even if we include the issue of how to best ensure that simulation, especially distributed simulation, contains enough detail to accurately represent these cues, these questions can be addressed by means of parametric analysis or similar techniques. Our intention in this paper is to provoke a more spirited debate, so we shall let our ambition reach farther.

In order to do this, and in keeping with the objectives stated above, we will broaden the definition of what we wish to consider as suppression. The US Army (FM-101-S-1, Operational Terms and Symbols p1-68) defines suppression as "direct and indirect fires ... brought to bear on enemy personnel, weapons or equipment to prevent effective fire on friendly forces." This, of course, raise the question of what is meant by "effective" fire, past studies (e.g., Torre, 1963) suggest that soldiers under fire become less accurate in their own fire, firing more rounds to less effect. This suggests that, by extension to other tasks, we consider suppression as degradation of performance (i.e., changes in the accuracy and rate of task accomplishment).

We will take suppression to include:

- potential degradation in performance which may be a result of the lingering effect of a single threat-associated environmental cue (especially small arms fire),
- potential degradation in performance which may be a result of the accumulated effect of many of such combat stressors.
- the alteration of current/planned actions as a result of these stressors.

To summarize: suppression is a disruptive response to an intrusive manifestation of a combat threat. In other words (as if any other words were needed!), the soldier perceives a threat which demands a reaction. If that reaction is in some way deleterious to the soldier's mission, then we describe this perception/reaction process as suppression.

4.2 Suppression: Perceptions/Reactions

Being ambitious, but not, however, entirely bereft of sense, we propose to limit consideration (at least at this time) to a small number of threat cues and associated reactions. In this paper we consider only stressors associated with kinetic energy weapons: small arms fire or indirect fire kinetic energy munitions. For the moment we will ignore natural environmental threats or other enemy weapons: e.g., nuclear, biological, chemical, directed energy. This is simply to keep our current analysis tractable; we anticipate that the methodology we are proposing could be adapted to handle such threats and indeed, in many cases they may manifest themselves in the same environmental cues (as listed below) provided by the threats we ARE considering.

4.2.1 Types of behavioral cues

Aural
- bullet/fragment impact/fly by
- small arms detonation/weapon muzzle sounds
- explosive detonation

Visual
- muzzle flash
- explosion
- obvious injury to a companion

These cues can be further characterized according to:

Intensity
duration - how long did it last?
number - how many cues might be lumped into one group, e.g. a burst of fire?
magnitude - how loud, how bright, how intrusive?

Frequency -what is the rate of arrival, or "inter-cue" time?

Proximity - how close was the cue event to the soldier being suppressed?

Immediacy - does the cue demand immediate, delayed, or cumulative reaction?

Ambiguity - does the cue indicate what the source of the threat is and where it's located?

Predictability - do the cues come from one area and/or in a discernible pattern

4.2.2 Types of soldier reaction

1) **immediate**, or **total, suppression** - the instinctive reaction to a threat as discussed above, manifested through one or more of:
   - freezing (complete cessation of movement),
   - assumption of extreme protective posture
   - panic
2) **partial suppression** - degradation in performance as a result of having experienced the threat or anticipation of the threat, manifested through:
   - increased time to detect, acquire, and engage targets
   - decreased accuracy in weapons fire, accompanied by increased volume (more shots/less hits)
   - decreased movement rates,
3) **defensive avoidance** - behaviors designed to reduce exposure to the threat, such as:
   - change of posture (e.g., standing to prone or kneeling)
   - increased use of cover,
   - alteration of routes of march, changed formations
   - panic and/or flight

Furthermore, while offensive actions may not be classically associated as a reaction to suppression, we would like to include them here because they represent a reaction to the threat which may delay or obviate original mission objectives; adding:

4) **offensive avoidance** - behaviors designed to reduce the threat, such as:
   - call for fire
   - initiation (or continuation) of suppressive fire against the threat.

Clearly, the above types are somewhat arbitrary and in fact overlap. This imposes consistency requirements on inter-related behaviors, e.g., prone soldiers must crawl, they can't run, so if we change posture from standing to prone we must degrade the soldier's rate of movement. At the same time we would probably also alter that soldier's movement mode to facilitate maximum use of cover, and change the soldier's mode of small arms fire as appropriate, with associated revisions in accuracy and rate of fire as well.

4.3 Translation from Cause to Effect

The suppression simulation paradigm as outlined so far is something like:

1) present the soldier with a behavioral cue (e.g., sound of bullet impact)
2) determine the appropriate response(s) to the cue (e.g., soldier changes from walking upright to crawling prone)
3) determine duration of response (e.g., soldier will resume an upright posture if one of the following happens:
   a) source of the bullet is eliminated
   b) soldier manages to move to a position shielded from source of bullet
   c) upon assurance of command authority that source of bullet is neutralized
   d) x minutes with no further indication of enemy activity occurs)
3) update soldier status and activity parameters as appropriate to that response (e.g., decrease soldier's speed and presented area)
4) calculate the soldier's task performance metrics subject to the updated parameters (e.g., increase time required for soldier to achieve next position objective, determine response of rest of soldier's squad)
5) As required, schedule new updates of soldier status.

There are a number of non-trivial details associated with this paradigm: how do we generate the behavioral cues, how do we handle the logistics of multiple simultaneous or near-simultaneous cues, how do we handle the interactions between multiple soldiers, and a myriad of other details of simulation event management. These details, while important, are not the central focus of this paper. The paramount questions for us are

1) how do we determine the appropriate soldier response to a specific cue (or in the more general case, to a complex sequence of such cues), and
2) how do we quantify the effects of that response on simulation parameters such as soldier movement rates and error budgets.

Possible approaches include:

1) **table look-up**: cues are matched to specific behaviors, with those behaviors specified in discrete increments to enumerate levels of response.
2) **aggregate measure**: integrate cues into a single measure of conflict, called ,for example, combat intensity, and make behavior parameters a dynamic function of this measure, for example applying a suppression percentage as a function of combat intensity.
3) **expert system hybrid** - expand the table of 1) into a set of situationally dependent rules to determine the appropriate kind of response, and use an aggregate measure as in 2) to determine the intensity of the response. This approach is similar to that described in the methodology for the US Army TRADOC Analysis Center's CASTFOREM model (Mackey et. al. 1994)

Approach 3) fits in neatly with the design philosophy of the IUSS. The IUSS simulates small unit and individual soldier performance as a networked series of tasks. These tasks follow the form of the Battlefield Operating Systems/Tasks (BOS-T) of the Army Training and Evaluation Program (ARTEP) manuals (US Army 1988). The ARTEPs describe literally any task a soldier may be called upon to perform, subject to the dictates of mission, enemy, terrain, troops, and time available (METT-T). Using the BOS-T/METT-T paradigm allows the IUSS to dynamically adjust task performance as appropriate to current conditions throughout the execution of a simulation scenario.

The IUSS already tracks many of the individual's psycho-physiological indicators and calculates response to battlefield stressors; we propose to devise an appropriate aggregate measure of combat intensity based on these indicators as adjusted to reflect additional response to suppression cues.

4.4 Features of Combat Intensity/Response Functions

Functions which measure combat intensity and associated responses should:

1) exhibit decay of effects over time, i.e., the longer the time since the observation of the cue, the less intense the effect (e.g., a decaying exponential)
2) allow for predominance of most intense effect, i.e., for multiple cues at the same time, the strongest one dominates
3) support re-enforcement by multiple effects over time, while allowing for some decrease in intensity due to familiarity
4) represent variable response to specific cues, i.e., to have a stochastic component. This component is essential for representing the tremendous variability in response from one individual to another, and indeed, the variability in response which could be exhibited by a single individual observed at different times.

Ideally these functions would support definition of a level of effects varying from a modest increase in an individual's caution, all the way to the ultimate suppression: complete deterioration of the will to fight and capitulation or abandonment of the battlefield to the aggressor force. As one side in the conflict begins to achieve success at suppression, the combat intensity function should represent the preponderance of fire by that side, and the associated increasing level of suppression of the opposing force.

5. Implementation and Application

Our initial implementation of a suppression simulation is intended to address the most obvious aspects of suppression. Briefly, we want individuals to duck when they're shot at, to proceed very cautiously if they're under fire, and to proceed with some degree of care if there is a possibility of enemy contact.

We propose a paradigm whereby environmental cues, such as small arms fire, will eventually be translated into quantitative effects on the

parameters which dictate the outcome of individual task processes. The IUSS, or any task/process model of behavior, must translate the effects of stressors such as suppressive fire into two fundamental measures of task performance: rate and the accuracy. The paradigm to achieve this translation is:

Given an activity
  Given a cue
    Given a context (cue and behavior history)
      Determine the probability distribution of suppressive effects of given intensities and durations
      Make a Monte Carlo draw against this distribution and apply the effects to the parameters which specify the accuracy and/or rate of the given activity.

To achieve our goals, while limiting our first cut implementation to a tractable level, we will limit consideration of suppressed behaviors to: 1) movement and 2) small arms fire. We will consider as suppression cues: 1) friendly force casualties, 2) enemy fire, and 3) friendly fire. The immediate response of the individual will be described in terms of changes of posture and use of cover, which will in turn affect rate of movement and accuracy of fire. Depending on the situational context and results of a Monte Carlo draw, the individual may increase or decrease rate of fire, may alter choice of targets and/or type of fire (single shot vs. burst, aimed fire vs. pointed or area fire), and may also advance, retreat, or freeze in position. Another Monte Carlo draw will determine the maximum duration of these responses to the suppression cues, i.e. the time at which behavior will return to pre-cue levels, assuming no additional cues or changes in task status.

We will develop a **SIMPLE** context-dependent expert system to determine the soldier's response to these behaviors and cues. Context will be established by examining cue history in terms of rate, intensity, and proximity. An additional important context consideration is the extent to which the individual can take effective action against the threat which produces the cues, i.e., if the individual can determine the source of the threat and has a course of action against that source.

At present we are developing the exact nature of the expert system rule set: the variables and their values which will define Monte Carlo draw parameters, and the appropriate levels of cue rate, proximity, and intensity to trigger suppression behaviors (e.g., how close does a bullet have to come to induce full defilade, how many casualties in how short a time will produce what probability of retreat, etc.?). As stated up front, one objective of this paper is to solicit the support of the community in this development process; the reader is invited to contact any and all of the authors for this purpose.

6. Future Directions

It is our intention to implement into the IUSS the simple expert system approach discussed above. By the end of FY95, we plan on using the IUSS and this simulation of suppression to explore issues relating to the lethality and survivability of the Twenty First Century Land Warrior. As discussed above, however, to assess the full potential of the 21 CLW will require examination of the synergistic effects of other capabilities, most notably command and control. We believe that the expert system approach we have outlined will lend itself to simulation of these capabilities and their contributions to mitigating the effects of enemy's attempts at suppression, through situational awareness, ability of units to respond for quickly and more appropriately to their leaders' decisions, and the ability of those leaders to integrate battlefield data to support their own decision processes. Furthermore, the context-sensitive nature of the expert system approach will permit consideration of the so-called "soft" factors: morale, training, leadership, national will, and others. These factors are important not only for estimating WHAT the effects of suppression might be, but also for determining HOW soldiers can most effectively adjust their actions to maximize suppression of the enemy and minimize suppression of their own forces.

7. Summary

We believe we have developed a framework adequate for beginning the discussion of first how to define suppression, and second how to

represent it in our models and simulations. We are currently working on the rule sets to describe the variable context-dependent reactions of the individual dismounted warrior to a limited set of perceived battlefield stressors. To the extent that those reactions are in some way deleterious to the soldier's mission, then we describe this perception/reaction process as suppression. We have begun the process of implementing this rule set paradigm into the Integrated Unit Simulation System, and are embarking on an iterative process of explaining our concepts, soliciting community and user feedback, and incorporating that feedback into the development.

8. Acknowledgments

The work described herein was supported primarily under contract DAAK60-94-1061 with the US Army Natick Research Development and Engineering Center. The opinions expressed are solely those of the authors and do not represent any official US Army position. The authors would also like to extend their appreciation to Dr. George Mastrioanni and Mr. John O'Keefe of the US Army Natick RD&EC and Daniel E. Mullally, Jr. of the Institute for Simulation and Training of the University of Central Florida for their ideas and support.

9. References

Haley, Richard L.; Shields, Joyce; Campbell, Crystal, C.; Godden, Gerald D.; Holter, Marvin R.; Laberge, Walter B.; Army Science Board 1991 Summer Study - "Soldier as a System"; Office of the Deputy Assistant for Research Development and Technology OASA(RDA), Attn: SARD-ZT, Washington D.C. 20310-0103; December 1991

Mackey, Douglas C.; Dixon David S.; Jensen, Karl G.; Loncarich, Thomas C.; Swaim, Jerry T.; CASTFOREM: Methodologies; US Army TRADOC Analysis Center - White Sands; White Sands Missile Range, NM; 88002-5502; April 1994; TRAC-WSMR-TR-94-010

Middleton, Victor E.; Integrated Unit and Soldier System Survivability and Effectiveness Evaluation Roadmap; US Army Natick Research Development & Engineering Center; Natick, MA; January 1992; Natick/TR-92/022L

Middleton, Victor E. and O'Keefe, John; *The Integrated Unit Simulation System: Quantifying Individual and Small Unit Mission Effectiveness on the Battlefield of the Future*; Proceedings of the 1993 Winter Simulation Conference; pp1043-1047; December 1993; Los Angeles, CA; G.W. Evans, M. Mollaghasemi, E.C. Russel, W.E. Biles (ed.s); ISBN 0-7803-1380-1 (softbound), 0-7803-1382-8 (microfiche); IEEE Catalog Number 93CH3338-1, Library of Congress Number 87-654182

Torre, James P.; Human Factors Affecting Rifle Accuracy in Automatic and Semiautomatic Fire; US Army Human Engineering Laboratories; Aberdeen Proving Ground, MD; May 1963; Technical Memorandum 11-63

US Army ARTEP 7-8-MTP, Mission Training Plan for the Infantry Rifle Platoon and Squad, dated September 1988.

US Army Materiel Command; Proceedings of the Soldier as a System Symposium/ Exposition; held at Hyatt Regency, Crystal City, Arlington, VA 22202; 30 June - 1 July 1992

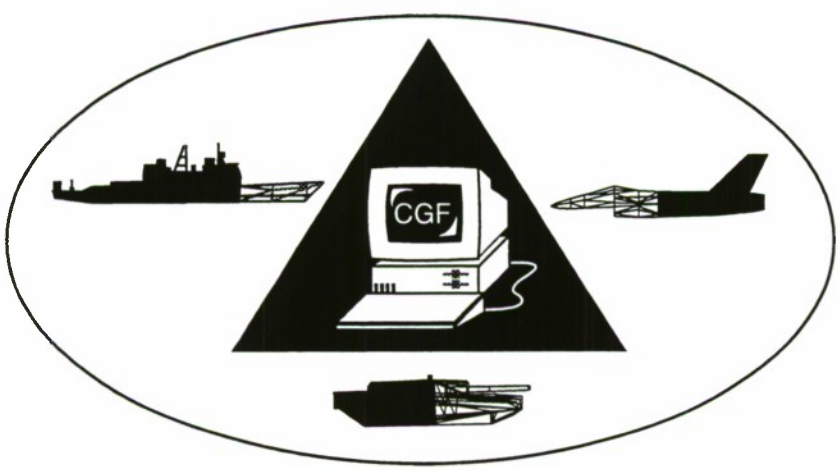
10. Author Biographies

Victor E. Middleton is a Senior Operations Research Analyst under contract to Simulation Technologies, Inc. of Dayton, OH, supporting the US Army Natick Research, Development & Engineering Center. He is responsible for the development of simulation algorithms and architectures for the Integrated Unit Simulation System. He has over 15 years experience in developing, implementing and applying mathematical models and simulations to a wide range of military and civilian studies and analyses. Mr. Middleton earned an MS degree in mathematics from Michigan Technological University in 1975 and an MS in applied mathematics from Michigan State University in 1978.

W.M. (Chris) Christenson is a Research Staff Member of the Institute for Defense Analyses, where he is the Coordinator, Janus Activities and an active participant across a broad spectrum of combat modeling and simulation activities. He served in the US Army from 1957 to 1983 and is a graduate of the US Military Academy (BS Engineering, 1957), the US Army General Staff

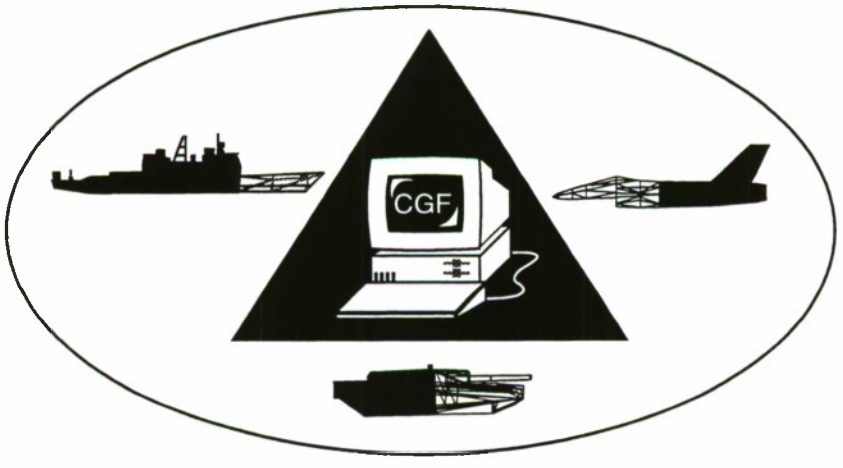
College, British Staff College, National War College, National Defense University, and George Washington University (MPA, 1978).

John D'Errico is an Operations Research Analyst in the Dismounted Battlespace Battle Lab, fort Benning Georgia. He has 12 years of experience in combat developments force-on-force models and simulations. Mr. D'Errico has a BS and a year's post graduate study in mathematics. He served 20 years active duty with the US Army and commanded an infantry company in the Vietnam War.


CGF

Session 10a: Architecture

Davies, DSTO
Gagné, IntelAgent R&D
Kwak, Loral ADS


CGF

Representing Role-Based Agents Using Coloured Petri Nets

Fred D.J. Bowden, Mike Davies and John M. Dunn

Information Technology Division
Defence Science and Technology Organisation, Salisbury
PO Box 1500,
Salisbury, South Australia 5108

1. Abstract

To assist in effectiveness studies of Command, Control, Communication and Intelligence (C3I) systems in the Australian Defence Force the Defence Science and Technology Organisation is currently developing the Distributed Interactive C3I Effectiveness simulation. A key part of this simulation is the use of computer generated forces. This paper describes one of the two types of computer generated forces which the simulation uses: artificial agents. These artificial agents are defined by a role-based structure, based on extended Petri nets, allowing for sequential and concurrent processes as well as top down and bottom up design techniques.

2. Introduction

The use of effective command, control, communication and intelligence (C3I) is very important in a military environment. To assist the Australian Defence Organisation, Information Technology Division of the Defence Science and Technology Organisation is developing computer software to enable the study of military C3I systems effectiveness. The nucleus of the software being produced, known as the Distributed Interactive C3I Effectiveness (DICE) simulation, will enable the study of C3I systems through distributed interactive simulation (Davies, Gabrisch 1995).

The DICE simulation environment is considered to be comprised of a number of interacting nodes. Each node represents a different entity in the central C3I system being modelled and the external environment about that system (Davies, Gabrisch 1995). All the nodes interact with each other by sending formatted textual messages along communication links. The message content depends on the nodes which are communicating. There are two different types of nodes in the DICE simulation: interactive players and computer generated forces (CGF).

Interactive players in the DICE simulation will generally be used to represent decision makers or commanders in the C3I system being studied. Interfaces are under development to allow the players to interact with the other nodes in the DICE simulation regardless of whether they are interactive players or CGF.

CGF in the DICE simulation are used to represent a variety of nodes, or cells within nodes, of the C3I system being studied and also entities external to the C3I system. C3I system entities can include single or groups of decision makers, centres of information filtering and processing, or combinations of these. External entities might include sensors, weapon systems, forces and also representation of the overall tactical-level battle or operation. Each CGF will have a unique structure that must be represented in the simulation. There are two types of CGF: peripheral units and agents.

In some cases there may be existing models and simulations that represent a given aspect of the system being studied, for example a battle simulation such as JANUS may be used to represent low level conflict between two forces. When models and simulations are incorporated into a given DICE scenario they are referred to as peripheral units. In general, peripheral units do not model C3I aspects of the system but are used to assist in measuring the effectiveness of the C3I system by helping form a representation of the overall military mission (Davies, Gabrisch 1995).

As is frequently the case with interactive simulations, there is often a need to use some form of CGF to represent different aspects of the system. In DICE each time a node cannot be represented by a peripheral unit or does not have a human player representing it, an artificial agent is designed to perform its task. The artificial agents in DICE are extended Petri net (PN) simulations and are

considered to be made up of a number of roles which are brought together to form the overall artificial agent structure. Each role may also be constructed from a number of roles, thus forming a hierarchical structure where the detail increases with depth. This document considers how these role-based artificial agents are designed and incorporated into the DICE simulation.

It should be noted that there will be times when studies will be carried out using a non-interactive simulation. In these cases the DICE simulation will be used and all the nodes will be either represented by peripheral units or artificial agents.

3. Requirements of Artificial Agents in the DICE Simulation

In considering the requirements of artificial agents it must be remembered that DICE is an interactive simulation and so involves both real and artificial players. The inclusion of interactive players in the DICE simulation adds to the requirements of the artificial agents.

In the DICE simulation the two types of agents, real and artificial, must interact in a way which will make interactive players unable to determine whether they are communicating with a real or artificial decision maker. The need for seamless integration is internationally recognised (Lewis 1994; Cox et. al 1994) and is very important to maintain a realistic simulation of the real environment. Interactive players cannot be allowed to determine between which nodes are CGF and which are interactive players as this knowledge may prejudice how they treat the other nodes. To help address this problem a standard language based on formatted textual messages has been chosen for communication between nodes[1]. Agents need to be able to comprehend and communicate in this language. Interface environments need to be developed for both human and artificial players that enable communications in the chosen language.

The requirements on the design of artificial agents in the DICE simulation are not unique to this project, other researchers have also seen some of them as important requirements of the CGF they are designing (Lewis 1994; Lankester et. al. 1994; Laird et. al. 1994; Cox et. al. 1994). It should be noted that research into the functions and interactions of real commanders is required in the development of artificial agents and the interfaces used by real players and peripheral units. This document concentrates on the design requirements of artificial agents.

3.1 Basic Artificial Agent Structure

Initially, the artificial agents to be designed are essentially rule-based; more sophisticated representations may be adopted as the field of artificial intelligence is researched and practical benefits determined. It has been expressed that current machine learning techniques are not sophisticated enough to deal with a problem this complicated (Cox et. al. 1994).

The fact that the artificial agents are rule-based, leads to there being a natural break down of their processes into smaller components. These components will be loosely termed "roles". A role is not necessarily the base element of the agent (as in the work by Levis 1993) it is more a function, which may in turn have many roles within it. This structure leads to a hierarchical design technique. By taking a hierarchical approach (Aronson 1994) the complexity at the level the designer is working at is reduced to a level which is relevant for the work being carried out. This method also supports the natural break down of complex systems into simpler ones. This means the artificial agent roles can be designed initially and then brought together to form the overall artificial agent, leading to a bottom up design approach. Alternatively the designer can initially sketch out the basic functions of the artificial agent and then add detail by enhancing its roles independently, leading to a top down modelling approach. The artificial agent being modelled and the information about the artificial agent will determine which method is most appropriate for the task at hand, making it necessary to allow either approach. This method also makes changes to the artificial agent easier as they will only involve change to a limited number of roles, keeping the rest of the agent unchanged.

C3I networks involve many different systems working both concurrently and consecutively towards the same main objective. This is also true for each of the nodes in the system. Thus, an artificial agent may involve events which occur either in series or in parallel. This is an important feature of any decision making process and so must be reflected in the artificial agent design. To some extent this is dealt with by taking the role approach, where roles are

arranged according to the way their related functions occur in the real system.

3.2 Explanation and Analysis Capability

Adequate credibility and realism in artificial agents is essential; judgement of such qualities can generally be best made by military domain experts. The underlying assumptions and characteristics of artificial agents need to be conveyable in a form easily understandable to a military expert. Such an expert should be able to interrogate features of the agents and to form some judgement on the realism of that artificial agent compared with the real-world system that the artificial agent represents. Having military endorsement of the assumptions used in such representations is a vital prerequisite to any C3I system study. The ability to have artificial agents, for example, explain their actions in an easily understandable form, is recognised as a very important requirement in the CGF arena (Cox et. al. 1994; Lewis 1994).

4. The DICE Simulation Artificial Agent Representation

Having defined the requirements of the artificial agent design in the DICE simulation it was then necessary to find an efficient method of modelling artificial agents with these features. Many different methods were considered, some of which are described below, before the decision on the use of extended Petri nets (PN) was made. The use of CGF in computer simulations is wide spread, however, this discussion is restricted to those methods taken by researchers who are designing CGF with similar structures to that of the artificial agents in the DICE simulation.

4.1 Finite State Machines and State Transition Diagrams

The main problem with using FSM is that they do not efficiently model concurrence and their representations can become very complex for large systems. The problem of complexity is partially overcome by using hierarchical FSM as in Cox et. al. (1994). However, the modelling of concurrence is very important in the artificial agents of DICE and this can not be represented easily by FSM. Other problems with the FSM approach are outlined in Harmon et. al. (1994).

4.2 Boolean and Fuzzy Logic

In Lankester et. al. (1994) the rules governing the actions of the agents are defined using boolean logic. As with the DICE agents the logic is set up in a hierarchical structure to try and keep the design controlled. This approach can be easily applied to the small system but becomes very cumbersome when it is used on more complex systems with large state spaces. This problem can be reduced by the use of fuzzy logic instead of boolean logic, as expressed by Parsons (1994), as this reduces the state space in most cases. However, regardless of whether boolean logic or fuzzy logic are used, artificial agents represented by either method are hard to change, and the implications of any changes are often very hard to determine.

4.3 Petri Nets

Petri nets (PN) have been used in the study of C3I systems by many researchers and even applied to the modelling of decision makers (Levis 19??). As such they have not been applied to the design of artificial agents in a DIS environment in the way required in DICE. However, PN have all the required features to model the operation of the DICE agents.

PN have a graphical modular representation that allows the artificial agent designer to represent the role logic in an easy and simple manner. Each role can be thought of as a PN which is then joined to other PN via common places and transitions, this allows each of the roles to be designed individually and then brought together. Alternatively a PN can be constructed outlining the artificial agent structure and then refined adding detail. Thus artificial agents can be designed in the same fashion as is described in section 2.1.

Due to the PN representation, PN are easy to modify and the effect of changes can be easily seen. An inherent feature of the PN structure is that it allows for the modelling of features such as synchronisation, concurrence and resource sharing, which are features the artificial agents in DICE will have. As with FSM, PN get very complicated and confusing when the complexity of the system increases. However, PN have been extended by adding elements such as coloured tokens and hierarchies which reduces the complexity of the graphical representation when representing large and/or complex systems.

In considering which method is best to model the DICE artificial agents it should be noted that a FSM is PN in which each transition has only one input arc and only one output arc. The fact that FSM are a subset of PN illustrates the advantage of using extended PN over FSM. Extended PN can also represent both boolean or fuzzy logic (Levis 19??). However, the main advantage that boolean or fuzzy logic have over the use of extended PN, in particular the boolean logic approach, is that it is easy for the layman to model and change the agents using this method. The problem with the logic approach is that once the system being represented gets large it is hard to make modifications and find errors, the graphical approach of extended PN does not have this problem.

In representing an agent the aim is to model a system that comprises discrete events occurring under defined conditions. It is this that makes extended PN the ideal tool for modelling such systems since this is what PN were designed for. It should be noted that the methods described above do not cover all the possible ways of modelling conditional event systems. However many of the other methods can be shown to be either equivalent to or subclasses of PN (Peterson 1977).

5. Simple Example of Role-Based Agent

The following sections introduce a simple example of a role-based agent and refer to this example to illustrate many of the concepts associated with the role-based methodology and other features. Artificial agent design and implementation will be achieved through use of the Petri net ANalysis Environment (PANE) that is currently being developed, which will automate many of the procedures that are discussed below. It should be noted that the example given here is purely being used as a way of demonstrating the role-based architecture described above and the reader should not be concerned about the mechanics of the PN.

5.1 Designing Roles

The role-based architecture of the DICE artificial agents is illustrated in the example given in Figure 1. The notation used for the PN in this example is that defined by Jensen (1990).

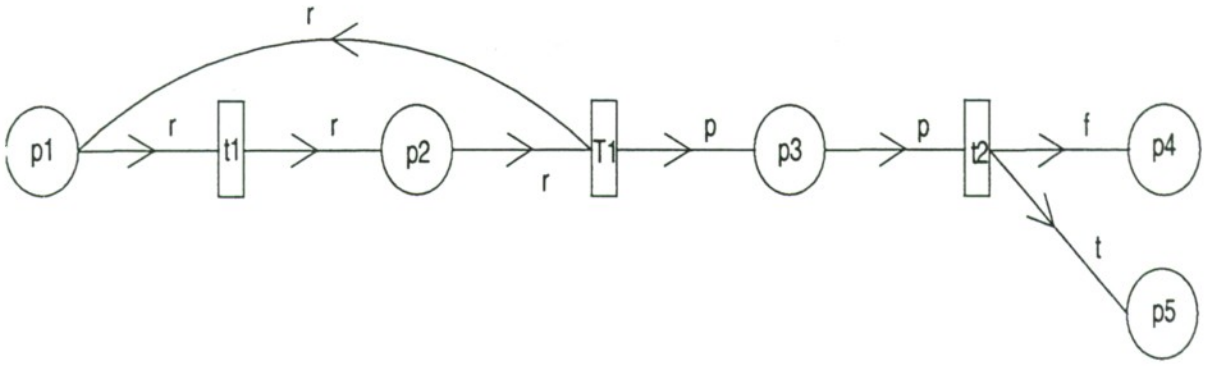
Consider the situation where a pilot requests permission from an air traffic control tower to taxi to a runway so that he/she can begin preparation for take off. If permission is given then the pilot will proceed along the taxiway, otherwise he waits for a period of time before submitting the request again. If a plane is taxying, then the taxiway is considered unavailable to other aircraft. On receiving a request the control tower must check the current weather conditions and then the availability of the taxiway before deciding if the pilot is cleared to taxi.

Figure 1 shows the role-based model representation of this system. In Figure 1 (a) the higher level model defining the roles as seen at the pilot level is shown. First he submits his request, which is represented in the PN by the firing of transition *t1* and the creation of an *r* token in place p2. Next the control tower responds, either approving or disapproving the request, *ie* transition *T1* fires. If the pilot is given permission to taxi then a *p* token is created in *p3*, otherwise an *r* token is created in *p1*. If the request is approved the pilot proceeds to taxi, transition *t2* fires on completion of the taxying placing an *f* token in *p4* to indicate the pilot has completed his taxi and a *t* token in *p5* signifying the availability of the taxiway.

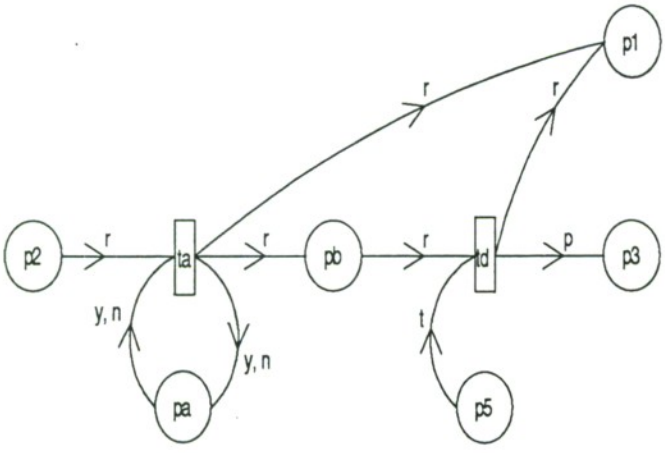
In Figure 1 (b) the details of the control tower are modelled. The extended PN shown here is a substitution transition to transition *T1* in the PN of Figure 1 (a). This model involves two different roles; the checking of the weather conditions (transitions *ta*) and taxiway (transitions *tb*). Also given in Figure 1 is a description of the physical meaning associated with the tokens of these nets. To make the associations between the two nets easily seen, like places and tokens have been given the same names.

The above example shows a top down approach to modelling. Initially the basic top level structure of the pilot net was defined and then the control tower role was modelled in more detail. An alternative bottom up design can be illustrated by considering the design of the control tower.

Figure 2 shows the two roles that exist in the control tower. Figure 2 (a) shows a general "checking of conditions" role and Figure 2 (b) shows a general "checking the availability of a resource" role. Having designed these two roles we can then bring them together to form the control tower modelled in Figure 1 (b). The two roles in Figure 2 could be regarded as existing in a library of roles from which they are selected and employed in constructing the control tower role. It is important to note that the lower level roles are originally general but become



(a) Pilot Net



(b) Control Tower Net

Token	Description
r	Signifies the submission of a request to taxi.
p	Signifies the control tower giving permission to taxi.
f	This token in place p4 indicates the pilot has completed taxying.
t	This token in place p5 implies that the taxiway is available.
y	This token in place pa implies the weather condition is satisfactory.
n	This token in pa implies the weather condition is unsatisfactory.

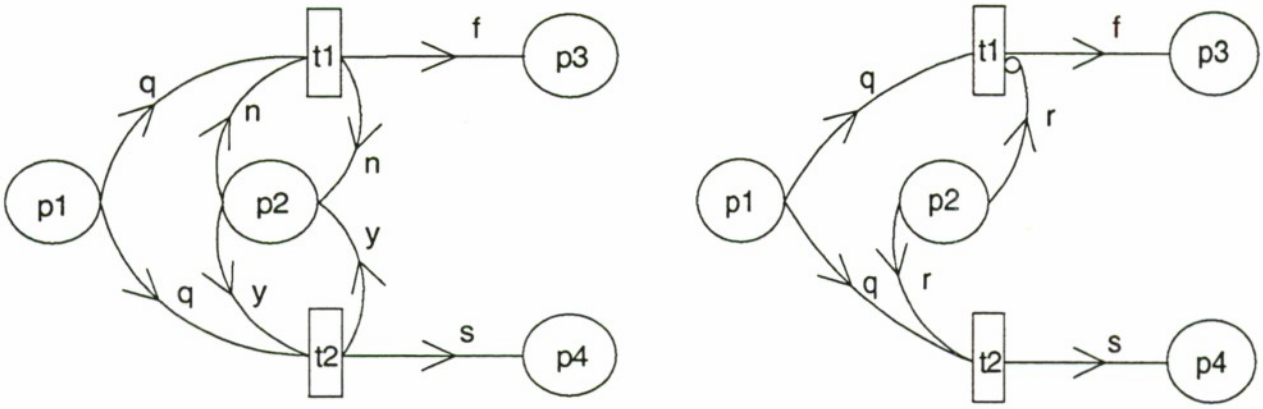
Figure 1: Aircraft Ground Control Example of Role-Based Agent

instantiated with a particular purpose when employed in the control tower role. The conditions to be checked become the weather and the resource being checked for availability becomes the taxiway. The PANE will allow for any combination of the above role-based design approaches as well as giving the designed systems the analysis and explanation capability required of them in the DICE project.

It should be noted that the control tower net represented here only makes up a very small part of what would be a complete control tower model. The portion shown is that activated by the particular problem being considered. This illustrates another nice feature of the PN role approach to agent design, only the relevant part of the PN model is activated when called upon.

5.2 Explanation and Analysis Capability

A PN explanation and analysis capability has been established using the declarative language Prolog. Prolog is a particularly suitable language for describing PN, where the basic connectivity characteristics can be regarded as a set of facts or clauses plus logical conditions. Perhaps the most powerful feature of Prolog, with regard to establishing an explanation and analysis capability is its search and multiple solution identification abilities. The analysis component interrogates a given Petri net according to user-specified goals or queries whilst the explanation capability abstracts and translates the query results such that they are presented in a form more easily understood to



(a) Check Conditions Role

(b) Check Resource Availability

Token	Description
q	Represents the submission of a query on a condition or the availability of a resource.
f	Indicates conditions are not satisfactory or the resource is not available.
s	Indicates conditions are satisfactory or the resource is available.
y	Corresponds to satisfactory conditions.
n	Corresponds to unsatisfactory conditions.
r	Represents the resource, its presence in p2 indicates that the taxiway is available.

Figure 2: Control Tower Roles

someone less versed in Petri net notation. The explanation capability is achieved through the use of declarative tags on the PN that give descriptions of the significance of a particular token residing at a particular place in the net; the firing of a particular transition mode; and summary information for an instantiated role.

A number of important queries were determined and are discussed below. The Prolog software automatically identifies any multiple, ie alternative, solutions to any query. The presence of roles allows explanations to be presented at a level appropriate to a user's query. A graphical user interface environment associated with this capability has been developed.

5.2.1 Reaction to a given situation

In a PN, a situation is described by a certain marking which, in raw Prolog form, might be [[p2,r,1],[pa,n,1]]. The PN explanation capability allows this situation to be expressed in the form:

['Permission to taxi requested',
'ct: Weather not suitable for taxi']

and it is in this form that a user can specify a situation. Consider specifying this situation and indicating that the level of abstraction required (the *target net*) is at the Pilot net. A query for the reaction to this situation would return an outcome of:

['Control Tower determined that permission should not be granted for taxi']

The mode that has actually fired in response to this goal is one associated with the Check Conditions net which checked the conditions (the weather) and returned in the negative. However, abstraction was used through the Control Tower net to the Pilot net resulting in the summary information above.

5.2.2 Sequence of Actions Resulting From a Given Situation

This clause generates an outcome consisting of a sequence of actions that occurs in response to a stated initial situation and resulting in some fully or partially defined final situation.

For example, with a target net of Pilot, if the initial situation were:

[' Permission to taxi requested',
'ct: Weather suitable for taxi',
'ct: Taxiway available']

and the final situation were any situation which contains the state:

['At runway']

then the response to this query would be the outcome:

[['Control Tower determined that permission should be granted for taxi'],
['Taxied to runway']].

Changing the target net to the Control Tower level would result in the outcome:

[['ct: Determined that weather is OK for taxi'],
['ct: Determined that taxiway is available for taxi']]

which indicates the same outcome but in terms relevant to the Ground Control net.

5.2.3 Explanation of Actions

This query gives the ability to select a particular action (or group of actions) and ask the question "Why did/would this action occur?". In the case of a role-based PN, actions will either be associated with the firing of regular transition modes or effective firings of substitution transitions. An explanation of why an action occurred or would occur is given, then, by the input requirements of the associated mode or a description of the mode summarised by a substitution transitions.

5.2.4 Sensitivity of Actions That Could Arise From a Given Situation

This query refers to taking a specified situation and requesting an illustration of the sensitivity of any actions (ie mode firings) to this situation. Processing of this query involves determining which firings are dependent on part or all of the given situation and ascertaining to what extent that situation would need to change in order for that firing to occur.

5.2.5 Actions That Could Result in a Given Situation

This query determines what groups of modes can fire such that their combined firings result exactly in a given situation. It is important to note that firing time considerations are not made here; the clause will return modes whose outputs upon firing cause a given situation, the modes do not necessarily fire simultaneously.

5.2.6 Actions That Could Contribute to a Given Situation

This query determines what actions could contribute to a given situation, ie what firings have output that is sensitive to the situation concerned. The result of any firings might not be exactly the required situation but could contribute to it. This approach is particularly important when multiple or repeated firings of a mode are required in order to produce a given situation.

6. Conclusions

This paper has outlined a role-based methodology for designing agents. This methodology allows for both top down and bottom up design, and uses extended PN to describe the roles. An explanation and analysis capability has been developed to accompany the PN technique. However, the explanation capability is only as good as the quality of the declarative tags attached to the PN and this will be the subject of further research as is addressing the transient and stochastic features that an agent can possess. Research is also being carried out to develop a PN extension designed exclusively to deal with the problems associated with role-based agent modelling.

7. References

Aronson, J. (1994) "The SimCore Tactics Representation and Specification Language", Proc. 4th CGF&BR Conf., Orlando, Florida.

Cox, A., Gibb, A., Page, I. (1994) "Army Training and CGFs - A UK Perspective", Proc. 4th CGF&BR Conf., Orlando, Florida

Davies, M., Gabrisch, C. (1995) "The Distributed Interactive C3I Effectiveness (DICE) simulation project: an overview", Proc. 5th CGF&BR Conf., Orlando, Florida.

Harmon S.Y., Yang S.C., Tseng D.Y. (1994) "Command and Control Simulation for Computer

Generated Forces", Proc. 4th CGF&BR Conf., Orlando, Florida

Jensen, K. (1990) "Coloured Petri Nets: A High Level Language for System Design and Analysis", LNCS 483, Advances in Petri Nets, pp. 342-416, Springer-Verlag.

Laird, J.E., Jones, R.M., Nielsen, P.E. (1994) "Coordinated Behavior of Computer Generated Forces in Tac-Air-Soar", Proc. 4th CGF&BR Conf., Orlando, Florida.

Lankester, H., Robinson, P. (1994) "GeKnoFlexe a Generic, flexible model for C3I", Proc. 4th CGF&BR Conf., Orlando, Florida.

Levis, A.H. (1993) "Colored Petri Net Model of Command and Control Nodes", Toward a Science of Command, Control, and Communications, Editor Carl Jones, Progress in Astronautics and Aeronautics, Volume 156, pp.181-192.

Lewis, J.W. (1994) "Agents that explain their own actions", Proc. 4th CGF&BR Conf., Orlando, Florida.

Parsons, J.D. (1994) "Using Fuzzy Logic Control Technology to simulate Human Decision-Making in Warfare Models", Proc. 4th CGF&BR Conf., Orlando, Florida.

Peterson, J.L. (1977) "Petri Nets", Computing Surveys, Vol. 9, No. 3, pp.223-252.

8. Biographies

Fred D. J. Bowden is a professional officer with the Defence Science and Technology Organisation in Australia. He has a Bachelor of Science degree in Physics and Mathematics and an Honours degree in Applied Mathematics. He is currently studying for a doctorate relating to the application of extended Petri nets to military C3I studies.
(Tel: +61 8 259 5205; Fax: +61 8 259 6781;
EMail: Fred.Bowden@dsto.defence.gov.au)

Mike Davies is a research scientist with the Defence Science and Technology Organisation in Australia. He has a Bachelor of Science (Honours) degree in Applied Mathematics and a PhD in Mathematical Modelling. His research interests are in the areas of military operation research, mathematical modelling and computer simulation.
(Tel: +61 8 259 6613; Fax: +61 8 259 6781;
EMail: Michael.Davies@dsto.defence.gov.au)

John M. Dunn is an information technology officer with the Defence Science and Technology Organisation in Australia. He has a Bachelor of Applied Science degree in Mathematics and Computing. He has worked at DSTO for six years, primarily involved in programming and computer support on PC and UNIX based systems.
(Tel: +61 8 259 5546; Fax: +61 8 259 6781;
EMail: John.M.Dunn@dsto.defence.gov.au)

Realistic Doctrinal Behaviors in Computer Generated Forces Through Plurality

Denis Gagné
IntelAgent R&D
1169 Notre Dame
Victoriaville, (Québec)
Canada, G6P 7L1

1. Abstract

Realistic doctrinal behavior in computer generated forces is a hard requirement to fulfill. We discuss the inherent diversity that exists in the assessment of realism of these forces and point out that building the right computer generated force for the task should be the goal aimed for. We argue that doctrinal realism can better be achieved via simulation entities consisting of multiple micro-agents and that the behavior requirements of the task should guide the level of abstraction selected. We present a system layer capable of supporting the ontology and an exploratory model of a simulation entity for an anti-submarine aircraft applying these concepts.

2. Introduction

Simulation training environments have always been predominant in the military training system. Advanced synthetic devices are now available to reproduce with high fidelity the characteristics of most existing weapon systems, providing ideal mediums for the transfer of procedural and technical skills. As simulation training technology evolved, the capability of interconnecting several manned simulators into a single simulation scenario was achieved. Today, network simulation environments, such as SIMNET and DIS, are central to the research and development efforts surrounding simulation training.

Network simulation environments have heightened the expectations about the resulting product of the training system. Increasingly complex readiness requirements are being specified which are aimed at attaining broader sets of tactical and judgment skills. Network simulation environments are now being proposed for a spectrum of combined arms operations ranging from mission training, mission preview, to mission rehearsal, as well as to analyze and evaluate strategies, tactics, doctrines and new weapon systems. Atop the requirement for realistic depiction of operational environments comes the requirement for the participation of plausible and believable opponent and supporting forces. Computer Generated Forces (CGFs) represent a cost effective solution to populate with varying numbers of simulation entities the operational environments of network simulations.

Scenarios to be engaged by CGFs vary greatly in complexity. In all cases, it is desired (if not required) that the CGF act/react in time conforming to the entity being emulated, and that the CGF behave rationally given the present operational situation and in accordance with the doctrine governing the force being simulated. Such requirements command the design of complex real time rational systems able to carry out a wide spectrum of tasks.

In this article, we describe current and ongoing research in the development of a general framework that can be applied to the design and implementation of CGFs. A discussion on doctrinal behaviors and abstraction levels points to a multi-agent ontology applicable to CGFs. We also describe an implementation of a system layer supporting that ontology. We conclude with an overview of an experimental model of a simulation entity based on the proposed ontology.

3. The Diversity of Realism

Realistic doctrinal behavior is desired of CGFs. This expectation generally applies to all types of CGFs, be they own, friendly or opposing forces. However, realism of a behavior is by nature qualitative which makes it an elusive goal to strive for [Har91]. There are at least two factors contributing to the elusive nature of this problem: the observer of the behavior and the definition of what constitutes realistic doctrinal behavior.

Realism of a behavior is in the eye of the observer and further depending on his standpoint and apprehension. Some observers are more critical or perceptive than others and will quickly recognize even the smallest unusual behavior in a participating simulation entity. An external unengaged observer focused on perceiving its environment will be more prompt to categorize an entity that deviates from familiar behavior, where as an engaged observer (or

active participant) only devoting peripheral attention to situation awareness will be more lenient toward delinquent behaviors (maybe not noticing). Probably the most influential factor is the observer's apprehension; if there is anticipation of participating CGFs in the scenario, the observer will subjectively be more critical of all behaviors.

Realistic doctrinal behavior is not all that "typical". Strategist and tacticians do not generally have unanimous agreement about the "right thing" to do in a given operational situation. Partly because individuals differ in their subjective appreciation of the present operational situation and because of the way each prioritize the tradeoff between conservative and aggressive course of actions. This in part is due to the fact that each rely on different partial information about the situation and each have their own set of values and beliefs with respect to what is best for the mission.

Conditions vary greatly across operational situations dictating different accomplishment strategies. Further, accomplishment of tactics occur under several adverse situational conditions, sometime leading to pointwise behaviors that are not in accordance with the doctrine. But it is the global behavior that is of interest. Not all combatants are expert tacticians or strategist, but theirs, the weapon systems they control, and the units they belong to, are the behaviors we want to capture. Further and most significant, flawless tactics and behavior bring about a sense of dealing with "super" entities which diminish the sense of suspended disbelief sought after.

Proponents of the "situated" behavior [Agr90, Bro91] make the argument that interaction and adaptation to the dynamic external demands of an environment is what makes a behavior realistic. For CGFs this is not enough, doctrinal behavior is also a key factor. What makes a behavior realistic in the context of CGFs is that the actions/reactions to the operational environment also generally follow a set of standard operating procedures that embodies the doctrine governing the force being simulated.

Realistic doctrinal behavior within the context of Computer Generated Forces can better be achieved via simulation entities consisting of a pandemonium of micro-agents [Hew73, Min85, Ten88]. Each micro-agent specializes in different and narrow aspects of operations. As they go about their tasks, these micro-agents confer with each other and form coalitions, producing collated, revised enhanced views of the raw data they take in. The process goes on ceaselessly. The information is under continuous revision so that at any point in time there are multiple views of information fragments at various stages of revision in various coalitions. These coalitions and their mechanism implement various cognitive processes and the desired doctrine.

Simulation entities can be individual micro-agents, several micro-agents, or even several other simulation entities aggregated. The recursive definition of the functional entity allows global control mechanisms to be applied at all levels of abstraction, from specific tasks to higher organizational levels. The higher levels of abstraction consist of coalitions of heterogeneous micro-agents. Through this plurality and heterogeneity, one can capture in a single simulation entity various conflicting priorities and globally incoherent knowledge, all of which are competing to suggest the next appropriate action for the given situation. Such simulation entities are more representative of real world situations where most complex military weapon systems are controlled by teams of crew members with diverse interpretations and intentions with respect to a given situation.

4. Selecting the Appropriate Abstraction

It is futile to try to design CGF entities that always do the "right thing" *per se*, instead we should concentrate on designing the right system for the task [Rus91]. Whether the CGF is needed for training, weapon system evaluation or developing doctrinal concepts, or whether the simulation entity is of the individual, team or aggregate level should determine the behavior requirements and direct the abstraction level to aim for.

A particular choice must be made for the number of micro-agents used to implement a particular abstraction. This choice can have a profound effect on the cognitive abilities and doctrinal correctness of the simulation entity as a whole, not withstanding the processing cost to maintain real time reaction. The ontology proposed favors knowledge distribution and functional autonomy which makes it ideal for distributed computing architectures, from transputer based machines to networks of computers.

Depending on the simulation being performed, the abstraction can be carried out at various levels. In a distributed type simulation, the micro-agents could emulate selected operational functionalities of crew members of the participating vehicles or vessels, where as in aggregate level wargaming, the micro-agents may be abstracted to key commander functions of a battalion or any other relevant military unit. Note that the recursive nature in the definition of a participating entity allows one to abstract to any desired level. The decision of the appropriate level to

model is guided by the amount of detailed behavior desired and the processing power available to maintain real time.

Analyzing and defining the appropriate abstraction for CGF entity is akin to the analysis and definition phases of the engineering lifecycle. It consists in identifying the required functionalities and behavior of the simulation entity and attributing them to a series of micro-agents. It is a tedious process for which not many tools are available. We are presently working at defining a toolset based on visual programming to alleviate this task.

We have been doing some work in defining a general agent theory to be applied to the team or crew level simulation. It is important to note here that within the context of this work, we are interested in the operational functionality of the team and crew members and not their personalities and/or higher cognitive abilities. Central to our agent theory and architecture is our belief that the main desirable property of the agent lies toward the reactive end of the cognitive abilities spectrum. We thus advocate a centrally situated and reactive philosophy [Agr90, Bro91] augmented with goal directed behavior.

We identified the following generic functionalities as required of the individual agents participating in a team or crew simulation entity: a reactive layer, a deliberative layer, a self model, models of his acquaintances, and a communication layer. These are depicted in figure 1.

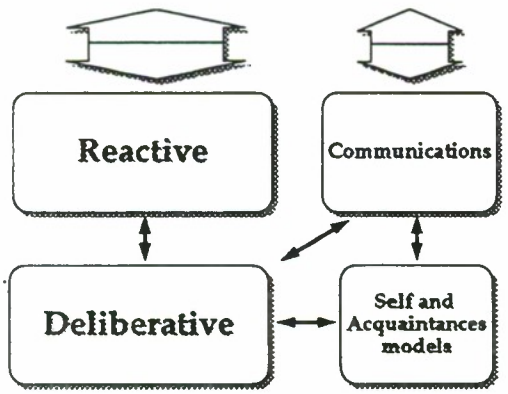


Figure 1: *Agent's Generic Functionality Layers*

Although depicted as modules in the figure, these only represent functional grouping and may actually be implemented quite differently. The reactive layer embodies the pre-compiled plans (behaviors) of the agent. It contains specialized micro-agents to address specific functions of the global agent (*e.g.* the routine operation of different sensors available to the crew member). The reactive layer provides the potential for real time performance. Rather than having to reason about which action should be taken next, the micro-agents of the reactive layer merely have to recognize their specific stimulus in a situation and carry out the associated action sequence.

The deliberative layer is where most reflective processing of the agent takes place. If an exception occurs, it is the task of the deliberative layer to reason about it and find a way out. The deliberative layer may also influence the reactive layer in its selection of the appropriate behavior. A degree of generality is achieved in the deliberative layer by using micro-agents with expertise in generic tasks. These generic tasks, which are meaningful in many different situations, when taken in context capture the domain specific information.

The self and acquaintance models consist of a collection of models capturing the agent's knowledge and beliefs of the present situation, and his beliefs and knowledge about other entities. We have reported on an ontology to capture such knowledge [Gag94b, Gag94d]. These models are a pre-requisite for any coordination of activities among the different participating members such as requesting information.

Agents require a communication facility in order to interact with others. This layer actually provides more services than just communication. The micro-agents of this layer actually carry out the transfer of information to the interested crew members as determined by the deliberative layer. Identification and composition of the appropriate request is the responsibility of the communication layer. Requests from others are similarly treated. Actual monitoring of the agent self status takes place within this layer.

Our present research focuses on perfecting and realizing this agent theory and architecture. Similar agent architecture have been proven quite effective in dealing with real-time environment [Wit92].

5. A Uniform System Layer

To support the above ontology for CGFs and agent theory and architecture, we generated a system layer called CLAP (C++ Library for Actor Programming.) CLAP is a uniform system layer implementing a variant of the ACTOR computational model [Agh86, Hew73]. The Actor model is a natural extension of the object-oriented paradigm, where each actor is an active, independent computational entity, communicating freely with other actors. CLAP implements the following concepts of the Actor

model: the notion of actor, behaviors, mailbox and parallelism at the actor level. Further, CLAP offers the extension of intra-actor parallelism to the model.

CLAP applications generally consist of many programs distributed over available processors, each executing as a task under the control of the CLAP run time environment. In CLAP, each actor is a member of a given task. The aforementioned actors are actually processes executing in parallel that may vary greatly in nature and size, from small simple processes to substantial software programs such as complete knowledge based systems. It is up to the programmer to determine how many actors are attributed to a given task (although, a large number of actors in a single task could mean the lost of potential parallelism in the application.) A scheduler controls the execution of processes inside the tasks. Each task possesses a message server that handles message reception for the actors in the task. Inter-processor message transmissions are handled via Remote Procedure Call (RPC) servers. External Data Representation (XDR) filters and type information are utilized for the encoding and decoding of these messages.

The present version of CLAP executes on a distributed heterogeneous environment consisting of SPARC workstations and Alex Informatique AVX machines which are transputer based distributed memory machines [Des93]. A port to HP and SGI workstations is in progress. An appreciable gain in power is achieved by distributing the actors over the network of processors making efficient use of the available processing power. With its exceptional modularity and grain size flexibility, our implementation allows complete interconnectivity at virtually any system level. Further, new actors can be created or destroyed dynamically at run time. These capabilities provide for a rather simple elaboration of multi-agent environments. We believe this approach to be promising in attaining the speedup and increased processing power required of large scale Distributed Artificial Intelligence (DAI) applications. CLAP is implemented in C++.

6. An Experimental Model

An initial experimental model was implemented for a CP-140 Aurora; a Canadian Forces anti-submarine aircraft [Gag93, Gag94a]. Rather than providing a single global behavioral model for the Aurora, the aircraft operational behavior is guided by a series of micro-agents each loosely capturing some operational functionalities of some of the crew members. A simple synthetic environment was implemented to exercise and validate the experimental model. The primary objective of the prototype leaned more towards demonstrating the feasibility of a multi-agent architecture for team or crew concept training [Gag95], than toward arriving at a model for Computer Generated Forces.

The selection of the abstraction level, and agent grain size, was based on the initial objective of providing the Aurora entity with a virtual crew for the purpose of crew concept training. The crew member abstraction level represents an intuitive functional decomposition for the operations of the aircraft. This abstraction allowed us to capitalize on the established standard operating procedures existing amongst crew members. As a result, there was no requirements to implement inter-agent negotiation protocol for the control or influence of aircraft operational events in this experimental model.

In this abstraction, the actual crew of ten was reduced to a virtual crew consisting of five members: the Pilot, the Tactical Navigator (TacNav), the Airborne Electronic System Operators (AESOP1 and AESOP2) and the Acoustic System Operator (ASO). Reassignment of crew duties involved piloting the aircraft, commanding the mission, and supervising and operating aircraft's detection instrumentation and weaponry. In this version of the experimental model, agents individually consist of a single expert system shell and a distinct knowledge base consisting of generic and operational knowledge. For this purpose the CLIPS expert system shell from NASA [Gir94] was ported and enhanced to execute in parallel on the distributed network [Gag94b]. The agents of this version of the prototype are monolithic and foremost reactive. The efforts of the next evolutionary step presently in progress concentrates on the elaboration and implementation of our recursive notion of agenthood to replace the present monolithic expert system shell architecture.

There are three main components to the prototype: the synthetic environment (simulated operational environment), the sensors and effectors, and the virtual crew. A situation awareness display interface provides a god's view of the theatre of operations while seemingly tapping the conversations taking place between the virtual crew members. The prototype is a multi-agent system executing on CLAP. The three components of the prototype are actually functional grouping of actors. All objects and entities of this prototype are CLAP actors: the simulation clock, the enemy vessels, the virtual crew members, etc. This uniformity of the systems layer provides a clean, intuitive, and flexible implementation. Each actor can individually be distributed on a separate processor of the network. CLAP supports the

dynamic creation and destruction of simulation actors, and manages all intra-actor communications. An earlier version of the prototype was implemented using a communication ring connecting the different agents to a simulation module[Gag94a].

A segment of the inter-actor communication network of the prototype is presented in *figure 2*. The figure only represents a snapshot since the network of participating actors changes dynamically at run time. The virtual crew's interpersonal communication network is depicted by a star in the figure. Note that the agents have no direct contact with simulation entities, except for the Aurora aircraft which they operate.

The perception the virtual crew has of the outside world (in this case the simulated operational environment) solely rely on instrument readings and the pilot's field of view. Individually, each virtual crew member is only allowed access to situational information normally available through sensors on board the aircraft or through interaction with other virtual crew members. This is a necessary condition to assure integrity of the global behaviour of the Aurora as an entity in a Computer Generated Force.

It must be stressed that the evolution of a mission in this implementation is completely non-deterministic (*i.e.* no canned scenarios or scripts), and solely based on the events taking place in the theater of operation and the decisions and actions taken by the virtual crew members (agents). An unrestricted number of enemy vessels can be dynamically introduced or removed from the simulation of the operational environment at run time. We consider this exogenous nature of our system to be a necessary condition, if we are to present computer generated forces as a valid experience gathering devices.

This experimental model has proven to be more interesting than traditional stochastic models because not only does it provide the non-deterministic characteristics desired but it also assures adherence to the overall doctrine by having individual virtual crew members carrying out doctrine correct operations in answer to the present situation. The prototype has not been evaluated other than subjectively by experienced operators. They were convinced enough by the system to support further exploration of this approach. Short term goals include embedding the Aurora simulation entity in a better operational environment simulation model.

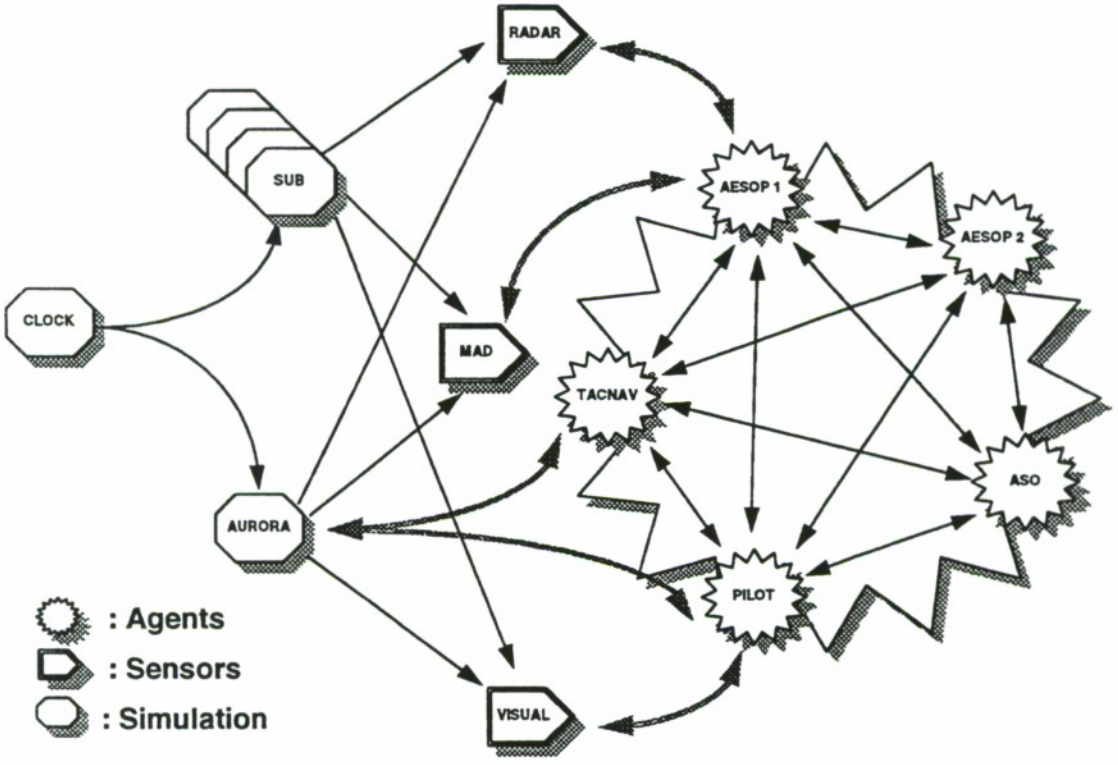


Figure 2: *Network of Actors*

7. Summary and Conclusions

The dream of an all encompassing, *omniscient* CGF entity is at best elusive. Designers of CGFs should concentrate on designing the right simulation entities for the task at hand.

We have argued that one should distribute the cognitive processes supporting doctrinal behavior of a simulation entity among specialized micro-agents of limited expertise. Through this plurality and heterogeneity, one can capture in a single simulation entity various conflicting priorities, and globally incoherent knowledge and interpretations of the present operational situation, all of which may be competing to suggest the next appropriate action. This approach best captures the diversity and non determinism that exist in pointwise behaviors of real operational entities while allowing to globally maintain realistic doctrinal behavior. We have described a system layer capable of supporting the design of such a simulation entity.

An agent theory and architecture was suggested for team and crew level simulation. This agent theory and architecture is the subject of further exploration in a crew concept training project. Finally, an experimental model of an anti-submarine aircraft was presented. Rather than providing a single behavioral model for the behavior for this simulation entity, the aircraft is controlled by a series of micro-agents. Behavioral integrity of this simulation entity was preserved by only allowing access to information normally available in a real operational situation.

We believe this general approach to be promising in attaining realistic doctrinal behavior for Computer Generated Forces and are presently working at the next generation of simulation entities based on these ideas.

8. Acknowledgment

The prototypes reported herein are the results of efforts from many people and where for most developed while at the Collège Militaire Royal (CMR) de Saint-Jean. I would like to acknowledge the support of Jocelyn Desbiens and his team at CMR without whom none of this would be. I also would like to thank André Trudel and Alain Dubreuil for their comments on earlier draft.

9. References

[Agh86]Agha, G., *ACTORS: A Model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge, Massachusetts, 1986.

[Bro91] Brooks, R., "Intelligence without Representation" In *Artificial Intelligence*, 47:139-159, 1991.

[Des93] Desbiens, J., Toulouse, M., & Gagné, D., "CLAP: Une implémentation du modèle Acteur sur réseau hétérogène." In *Proceedings of the Department of National Defence (DND) Workshop on Knowledge Based Systems/Robotics.* Ottawa, Ontario, 1993.

[Gag95]Gagné, D., "Training the Crew Concept via Multi-Agent Systems." *In Proceedings of the International Training Equipment Conference (ITEC95).* The Hague, Netherlands, 1995.

[Gag94a]Gagné, D., Desbiens, J., & Nault, G., "A Multi-Agent System Simulating Crew Interaction in a Military Aircraft." *In Proceedings of the Second World Congress on Expert Systems.* Estoril, Portugal, 1994.

[Gag94b] Gagné, D., & Trudel, A., "A Multi-Agent Temporal Structure: Preliminary Results." In *Proceedings of the Canadian Workshop on Distributed Artificial Intelligence (CWDAI),* D. Gagné (Ed.), Banff, Alberta, 1994.

[Gag94c] Gagné, D., & Garant, A., "DAI-CLIPS: Distributed, Asynchronous, Interacting CLIPS." *In Proceedings of the Third CLIPS Conference.* NASA: Lyndon B. Johnson, Houston, Texas, 1994.

[Gag94d] Gagné, D., Pang, W., & Trudel, A., "A Spatio-Temporal Logic for 2D Multi-Agent Problem Domains." IntelAgent Technical Report, 1994. Submitted *to First International Conference on Multi-Agent Systems (ICMAS),*

[Gag93] Gagné, D., Nault, G., Garant, A., & Desbiens, J., "Aurora: A Multi-Agent Prototype Modelling Crew Interpersonal Communication Network." In *Proceedings of the Department of National Defence (DND) Workshop on Knowledge Based Systems/Robotics.* Ottawa, Ontario, 1993.

[Gir93] Giarratano, J. & Riley, G., *Expert Systems: Principles and Programming.* PWS Publishing, Boston, Maine, 1993.

[Ham91] Hammond, S., Yang, S., Howard, M., & Tseng D., "A Behavior-Based SAFOR and its Preliminary Evaluation." In Proceedings

of the 2nd Behavioral Representation and Computer Generated Forces Conference. Orlando, Florida, 1991.

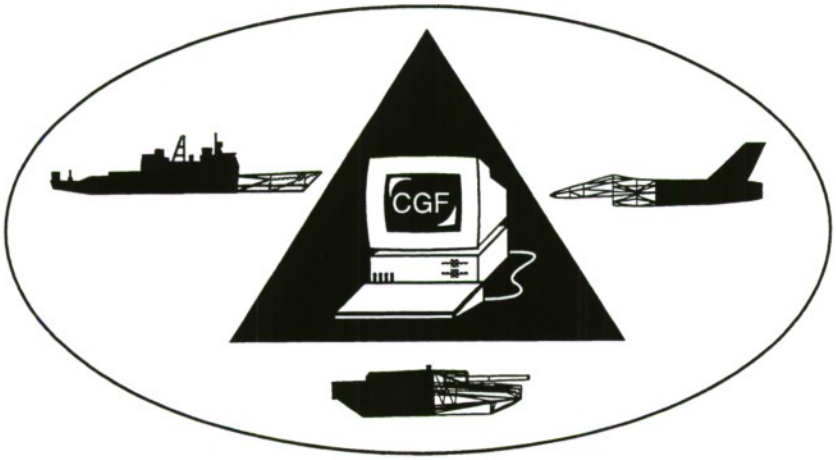
[Hew73] Hewit, C.E., Bishop, P. and Steiger, R., "A Universal Modular ACTOR Formalism for Intelligence." In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-73).* Stanford, California, 1973.

[Min85] Minsky, M., *The Society of the Mind.* The MIT Press, Cambridge, Massachusetts, 1985.

[Rus91] Russell, S., & Wefald, E., Do the Right Thing: Studies in Limited Rationality. The MIT Press Cambridge, Massachusetts, 1991.

[Ten88] Tenney, R. & Sendel, R., "Strategies for Distributed Decision Making." In *Readings in Distributed Artificial Intelligence*, Bond & Gasser (Eds.). Morgan Kaufmann, San Mateo, California, 1988.

[Wit92] Wittig, T. (Ed.), *ARCHON: an architecture for multi-agent systems.* Ellis Horwood, West Sussex, England, 1992.


CGF

A Comparison Study of Behavioral Representation Alternatives

Se-Hung Kwak
Loral Advanced Distributed Simulation
50 Moulton St., Cambridge, MA 02138
skwak@camb-lads.loral.com

1. Abstract

There are fundamentally two approaches to implement behaviors for a CGF computer system. One is a forward reasoning approach and the other is a backward reasoning approach. Even though the forward reasoning approach is commonly used to implement behaviors in CGF because of its initial intuitiveness and historical familiarity, the other alternative should be carefully examined. Because of a unique philosophy behind military doctrines; hierarchy, unambiguity, and purposefulness, a rather opportunistic forward reasoning approach is less suited than a goal directed backward reasoning approach. In this paper, these two approaches and representations are examined and compared with a specific military task example, "Reaction to Air Attack Drill" of US Army Tank Platoon.

2. Introduction

One of the ultimate goals of research in computer generated forces (CGF) is to create automated agents that behave as humans would on a real battle field. Although some CGF softwares, such as ModSAF, have been successful to implement realistic CGF behaviors in a simulated battle field, their behavioral implementations have been somewhat focused on low level echelon behaviors.

As climbing echelon hierarchy in military, the echelon's roles are getting detached from physical entities, such as tanks and helicopters. They are getting inclined to command and control rather than simple movement and shooting which are dominant roles (or behaviors) of a low level echelon. They spend most of time to reason about mission, enemy, terrain, troops, and time available (METT-T) to generate orders. Thus, their roles are behaviorally more complex than those of a lower echelon.

Currently, each CGF software utilizes its own behavioral implementation for CGF. For example, ModSAF uses Augmented Asynchronous Finite State Machine (AAFSM) to implement behaviors. However, when CGF software starts to simulate higher-level echelon behaviors, the current behavioral implementation methodology might not well suit for a larger scale future CGF.

In this paper, two behavioral representation alternatives will be compared; i.e., AAFSM, and Rational Behavior Model (RBM). AAFSM represents a data-driven forward reasoning behavioral approach, and RBM represents a backward reasoning behavioral approach. A data-driven forward reasoning approach tends to lead an exploratory implementation; i.e., designing a behavior from a start state to a goal state based on a given sequence of external inputs, while the latter backward approach usually guides to a structural top-down behavioral decomposition during its design stage. Thus, depending on targeted behaviors to be implemented one approach is expected to be better than the other. As an example, Reaction to Air Attack Drill behavior is chosen and implemented to compare their advantages and disadvantages, respectively. Especially easiness of converting published military Field Manual to both behavioral representations is discussed.

3. Behaviors and Behavioral Representations

3.1 Systems and Behaviors

A computer based system, such as a CGF and a robot control system, basically shares great commonalities with other biological or social systems. That is, as a system, it continuously manages internals and interacts with other external systems. A system may contain multiple sub-systems, while it also becomes a sub-system of other systems. Therefore, in this paper, a system is described as a recursive entity that has intra- and inter- interactions between sub-systems and external systems.

Interactions between systems are directly caused by interactions between behaviors in the systems. And the behaviors are entities capable of producing such interactions. There are two kinds of behaviors; i.e., *internal behavior* that is not observable from outside and interacts only internally, and *external behavior* that is externally observable and interacts with other external systems (including super-systems). The

external behavior is simply called *behavior* of a system because it produces readily observable behavioral results.

System interactions caused by a behavior do not spontaneously and aimlessly occur without a purpose. They are (especially in a man-made system) controlled by a sound logic that is either deliberately or optimistically[1] implemented to reflect the purpose. Thus, a behavior should have means to include such behavioral control logic. The behavior itself cannot exist in a vacuum. It needs an embodiment where it can place "memories" needed to maintain individualistic nature of a behavior. Without the embodiment, multiple behaviors with the same behavioral control logic would produce indistinguishably identical behavioral results regardless their distinct situational histories. Finally, a behavior has to interface to other behaviors through either transformation or actuation. Though the last factor seems purely an implementational issue, without the interfacing activity, a behavior is not able to interact to other behaviors. Consequently, the behavior, in this paper, has three components; i.e., a behavioral control logic, an embodiment, and interfacing means.

A *system behavior* manages all the system internals to produce an external system interaction. Specifically, it controls the overall operations of internal behaviors with the behavior control logic, records those operational results in the embodied portion, and interfaces to other systems to produce tangible system interactions. It can be dynamically delegated from the internal behaviors, but a statically assigned system behavior is chosen in this paper because of simplicity and easiness to maintain the whole system stability.

The internal behaviors directly come from the sub-systems, and they are manipulated by the system behavior to constitute external system interactions. Therefore, the system behavior is the external behavior of a system. Even though the system behavior mainly combines all available internal behaviors, it is not a simple combinatorial combination of them because of the embodiment and the behavioral control logic of the system behavior. Thus, the system behavior (the external behavior) is more powerful than a mere collection of those of the sub-systems.

Consequently, building a system becomes constructing a system behavior that manages the internal behaviors and including sub-systems that can provide necessary internal behaviors. If some of the behaviors are not readily available, then sub-systems that can provide the missing behaviors have to be newly built. This system building process recursively continues until all needed behaviors are included[2]. There are many proposed behavior models and related development/implementation tools. Most of them (Kwak 1990, Kwak 1992, Kwak 1993, Scholz 1993, Byrnes 1993, Loral 1995, Rosenbloom, 1993) implicitly or explicitly support this system building approach.

In this paper, those behavior models and tools are classified into two categories depending on the behavioral control part because two extreme representations are available; i.e., forward and backward reasoning representations (Jackson 1990). In a pure predicate calculus sense, two representations are totally equivalent. However, when either of the representations is used to implement the behavioral control logic that operates in a context rich environment, the equivalence issue is not as simple as a syntactic switch from one form to the other. The order of logic proving sequence becomes significant because of side-effects caused by the logic proving process. If the logic proving sequences vary from one implementation to another implementation, then there is no way to guarantee that the sequences of observable behavioral results caused by the logical proving process in both implementations are identical, even though they are logically equivalent. This observation leads to a special equivalence called *behavioral equivalence* (Kwak 1993). An existence of behaviorally equivalent pair in both representations was found and reported in (Kwak 1993). A similar finding in the context of factory assembly control was also reported in (Homem 1990).

3.2 Forward and Backward Reasoning Representations

The forward reasoning means a series of possible traversals from the start to a goal, while the other reasoning means a recursive goal decomposition; that is, starting from the goal, goals are recursively decomposed into simpler sub-goals. This process stops when simplest primitive goals are encountered, which can be directly accomplished by the sub-systems. Depending on the problem space of a behavioral control, either or both approaches might be susceptible to a combinatorial explosion. However, one approach is usually better off than the other approach.

[1]Some behavioral implementations are based on optimistic expectations (Brooks 1986)

[2]This is a top down approach. A bottom up construction is equally possible.

Practically speaking, the recognized problem space of a behavioral control continuously evolves from the initially recognized space because of knowledge accumulation (experience) of a designer/implementer about the problem[3]. At the beginning of the behavioral implementation phase, his understanding of the control problem is minimal. From the start state, he can experiment various operations to reach the goal or at least to try to move toward the goal. If such a move is apparently closer to the goal based on his simplistic evaluation model, a new state is introduced to maintain the move and to simplify future moves. Unfortunately, the successive movements may lead to a trap (local mini/max) state without any further improvement due to the error of his simplistic evaluation model. However, a series of such trials and errors will (most of cases) eventually lead to a path (or multiple paths) to the goal. At this point, the collection of all the paths explored forms a recognized problem space that should be much limited compared to the complete problem space.

If the goal is reached, then the whole problem can be viewed from the goal to the start. At this point, he starts to approach the problem by discovering goal and sub-goal relationships. Such investigations lead to form a hierarchically organized goal-subgoal tree. During this process, all (practically) possible branches (subgoals) can be systematically identified because of the top-down approach. Frequently, the complete problem space does not need to be identified. A certain subgoal branch may be ignored. If so, the branch is terminated by a simple default subgoal. When such tree construction is completed, an adequately shaped behavioral control space is available. This is possible because of the gained knowledge, i.e., experience.

The degree of understanding of the problem space leads to a selection of a behavioral representation. If the adequately shaped goal-subgoal tree is not available -- a behavioral control problem is partially understood or unstructurally described -- then a forward reasoning representation and implementation will be a better choice. If such a goal-subgoal tree is available, then a backward reasoning representation will be the choice because a paradigm match between an explored problem space and a representation results in greatly reduced development time and efforts.

[3]We, human beings, are not fully capable to grasp the entire problem space of a realistically sized behavioral control problem. We intentionally or unintentionally simplify or limit the size and shape of a behavior control problem in order to effectively handle the problem with our limited resources.

4. AAFSM & RBM

4.1 AAFSM (Asynchronous Augmented Finite State Machine)

Asynchronous Augmented Finite State Machine (AAFSM) (Loral 1995) is chosen to represent the forward reasoning representation. Though AAFSM is highly specialized for ModSAF to facilitate behavior implementation in ModSAF, it is basically a Finite State Machine (FSM), which defines states and state transitions.

Specifically, in ModSAF, behaviors are implemented by tasks, and the tasks are implemented by AAFSM's. They are asynchornous because they change states in response to events in the simulated environment. However, if needed, ModSAF system clock, which can be simulated or synchronized to real-time clock, can trigger the state transitions. Thus, both asynchronous and synchronous state transitions can be implemented with AAFSM's. They are also augmented state machines because they use many variables, predicate functions, and procedures to handle variables other than their state variables.

AAFSM implements behavior control logic with its FSM portion, and the augmented part provides the behavioral embodiment and the behavioral interfaces. Thus, AAFSM basically provides three components needed for implementing behaviors, which are described in the previous section. Recursiveness is indirectly supported in conjunction with the rest of the ModSAF architecture, such as tasks and task frames.

The ModSAF environment also provides a tool to construct AAFSM, which facilitates a programmer to define an AAFSM. Rather than constructing an AAFSM with an ordinary programming language, such as C, he can write an AAFSM with the predefined AAFSM macro facility. Then the AAFSM code is translated by a preprocessor utility called "fsm2ch". The fsm2ch utility is written in "awk" script and converts an AAFSM code into a C source code so that it can be compiled together with other ModSAF programs that are written in C.

The fsm2ch macro facility provides a syntax and commands needed to express state transitions, event declarations, criteria declarations, and task related commands (Loral 1995). The fsm2ch facility also allows a programmer to write C source code to complete an AAFSM. That is, operations in a state and state transition predicate functions are written in C programming language, but the state construction

and state transition themselves are expressed by the fsm2ch macro facility. Specifically, an AAFSM state starts with a grave character(`) and followed by the name of the state as shown in Fig. 1. After the name declaration, the definition of state follows. The operations triggered by ModSAF system clock are defined first in the state definition. Operations initiated by ModSAF "param" event follows. Optionally user defined events can be declared and operations related those events can be placed after param event operations. The param event is a ModSAF specific event that is triggered whenever there are changes in the inputs of the task containing the AAFSM. For example, if a route input to the task is newly updated, then C code defined in param section of a current state is executed as well as an ordinary system clock based operation also defined in the state. Figure 1 shows one of AAFSM state.

```
`
under_air_attack
`    tick
     if (time_to_monitor(private))
     {
        check_enemy_force_spotted(private, vehicle_id, unit_entry);
        check_unit_under_fire(vehicle_id, private, unit_entry);

        if (no_air_veh_for_long_time_p(vehicle_id, private, unit_entry))
        {
                send_contact_report(vehicle_id, private);
                private->time_last_monitor = 0;
                STOP Targeter
                ^monitoring;
        }

     }

`       params
```

Figure 1: AAFSM state

4.2 RBM (Rational Behavior Model)

Rational Behavior Model (RBM) (Kwak 1992, Byrnes 1993) is chosen to represent the backward reasoning representation. RBM is a multi-paradigm, multi-level intelligent control architecture. This architecture composed of three levels; i.e., Strategic, Tactical, and Execution levels. The behavioral control is performed by the Strategic level where behavioral control logic is located. The Strategic level governs the operation of the Tactical level. The Tactical level embodies behaviors by maintaining behavioral attributes for the system, which includes system memory and world-and-local memory models. The Tactical level also forms a representation of internal behaviors for the Strategic level. Finally, the actual behavioral interface to other behaviors is located in the Execution level. Therefore, RBM explicitly supports three components needed for implementing behaviors. Recursiveness is also directly supported by RBM through the behavioral abstraction of sub-system behaviors in the Tactical level. RBM recursively defines a system with the sub-systems and associated behaviors by encapsulating them in the Tactical level.

The backward reasoning nature of RBM comes from the Strategic level. This level is usually written in Prolog, which textually describes AND/OR Goal Tree. In other words, there is one-to-one direct correspondence between the Prolog code in the Strategic level and AND/OR Goal Tree (Kwak 1993, Byrnes 1993). Specifically, AND/OR Goal Tree is an extended version of AND/OR Tree by augmenting backtracking of Prolog. The Prolog code in the Strategic level is a limited version of Prolog by disallowing use of assertions. This means that there is no memory at the Strategic level other than the memory of the Prolog inference engine, which allows backtracking in the Prolog code. Additionally, the textual order in Prolog code determines a priority of the code execution. Likewise, there is a node priority among siblings under the same parent node in the AND/OR Goal Tree. The node priority and the Prolog textual priority are used as a conflict resolution scheme if equally possible subgoals are encountered. Basically there are two types of subgoals; ORed and ANDed subgoals to a parent goal. The priorities are equally applicable to both types of subgoals, but only the priorities among ORed subgoals are treated as being significant because of possible conflicts among the ORed subgoals. In a pure predicate logic, there is no priority among either ANDed or ORed subgoals.

Prolog code and its corresponding AND/OR Goal Graph are in Figure 2.

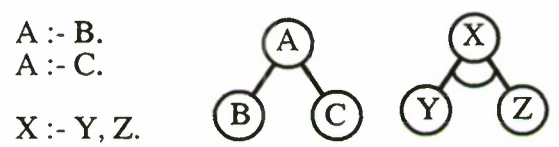


Figure 2: Prolog code and AND/OR Goal Graphs

There are a head and a body for each line of Prolog code, which is terminated by "." like a common English sentence. The head and body are separated by ":-". The left side of ":-" is a head and the other side represents a body. The meaning of ":-" is in order that a head is true, the body should be true, or if a body is true than the head is true too. For example, the first line of the Prolog code says: in order that "A" is true, "B" should be true. Or if "B" is true, then "A" is true too. There is a mechanism to present logical relationships; such as AND and OR.

The last line of the Prolog code shows an AND relationship, that is, in order that "X" is true, "Y" and "Z" should be true. An OR relationship is shown in the first and the second lines. In order for "A" to be true, "B" or "C" is true. A direct interpretation of the two lines is following: in order for "A" to be true, "B" should be true. If not, the "C" should be true. If not, there is no way to make "A" true. These Prolog constructs can be recursively applied.

An AND/OR goal tree has one-to-one correspondence to a RBM Prolog code. A node corresponds to a head of Prolog code, and the body becomes child nodes. An AND/OR goal tree is also able to show two relationships; i.e, AND and OR. The ANDed children are connected by an arc between branches, but the ORed children do not contain the arc. Again, these mechanisms can be recursively applied like the Prolog constructs. An AND/OR goal tree is able to provide immediate feels of the goal-subgoal relationships. However, its practical usefulness is rapidly degraded as the size of the tree increases. The intuitive graphical appeal is very much offset by a graphical complexity of a large AND/OR Goal Tree.

5. Reaction to Air Attack Drill Behavior

ModSAF's unit level react air task is a direct implementation of "Reaction to Air Attack Drill" of US Army Tank Platoon Field Manual (FM). In this section, the FM will be briefly discussed and followed by two implementations; i.e., AAFSM and RBM.

5.1 FM 17-15

FM 17-15 (Tank Platoon), pp. 3-21 - 3-23 (Army 1987), describes "Reactions to Air Attack Drill". The FM states "the tank platoon should conduct a passive defense against air attack", and describes:

> Alert the platoon. Air guards can alert the platoon using either of two techniques: announcing "CONTACT-BANDITS-(direction)" over the radio, or using hand-and-arm signals.
>
> Seek cover and concealment. When moving, tank units must immediately seek cover and concealment. If concealment is not available, moving tanks should stop. A motionless tank is harder to see than a moving tank. (If enemy aircraft detects the tanks and initiate an attack, the platoon leader announces "AIR ATTACK," and exposed tanks move at a 45-degree angle toward or away from the attacking aircraft.) Vehicle should maintain at least 100 meter intervals and avoid presenting a linear target in the direction of attack.
>
> Prepare to engage. Tank commanders and loaders should prepare to engage aircraft with a high volume of machine gun fire on order of the platoon leader. Since firing machine guns could reveal the location of concealed vehicles, the platoon leader must make sure the aircraft are attacking. The platoon leader may designate an aiming point for the platoon with a burst of tracers. Volume is the key to effectiveness. The tanks throw up a wall of fire and let the aircraft fly through it. The tank main gun can be used effectively against hovering enemy attack helicopters with a high probability of kill. Leaders should consider enemy attack helicopters as tank killers and should take actions to kill them before engaging less dangerous targets.
>
> Report. The platoon leader sends the commander a contact report. Example: "CONTACT-HELICOPTER-SOUTH".
>
> Enemy aircraft operate in flights of two, four, six, or more. After the first aircraft passes overhead, another may follow. Tanks should remain in covered and concealed positions for at least 60 seconds after the first aircraft leaves.

The above FM description about React to Air Attack Drill is by no means a complex task for trained tank platoon personnel because it is written for trained military men/women who have a common background knowledge about military operations and a human common sense reasoning capability. However, it cannot be directly used as a behavioral description for CGF. This limitation leads to an extra transformation to a structured and precise behavioral description. In this paper, portions of the above description are omitted because of functional limitations of ModSAF 1.0 at the time of the implementation. For example, ModSAF 1.0 did not support a cover and concealment seeking behavior against an aircraft. Consequently, the ModSAF version of React Air behavior becomes following:

Alert the platoon.

Avoid air attack.
If moving, then scatter.
If not moving, then do nothing (or keep stationary).

Under air attack.
If moving, move until having a platoon's proper defensive posture.
If not moving, start scattering.
If proper defensive posture (there are enough space between individual tanks), then counter attack.

Report.
If no air vehicles are spotted for at least 60 seconds.

The above description is very close to the ModSAF React Air behavior, but it cannot be directly a running computer program. We need further transformations for the following implementations.

5.2 AAFSM Implementation

For AAFSM implementation of React Air behavior in ModSAF, three states and three extra supporting states are introduced[4]. They are "monitoring", "avoid_attack", "under_air_attack" states, and "START", "END", "SUSPEND" states, respectively. Except the "START" state, performing the initialization for the AAFSM, the other two supporting states are just empty in this implementation. The three main states perform the necessary behavioral control. The AAFSM implementation is in Figure 3.

```
START
    PERIOD private->params->tick_period
    switch (state->state)
    {
    case monitoring:
        ^monitoring;
    case avoid_attack:
        ^avoid_attack;
     case under_air_attack:
        ^under_air_attack;
        break;
    case ended:
    default:
        ^monitoring;  wait for enemy contact
    }

monitoring
    tick
    if (time_to_monitor(private))
    {
        check_enemy_force_spotted(private, vehicle_id, unit_entry);
        check_unit_under_fire(vehicle_id, private, unit_entry);

        if ((private->under_fire) &&
            is_proper_defensive_posture(vehicle_id, private))
        {
            make_umxtravel_parameters_available(vehicle_id);
            SPAWN SELF Targeter
            ^under_air_attack;
        }

        if (is_moving(vehicle_id) && (private->under_fire))
        {
            make_umxtravel_parameters_available(vehicle_id);
            SPAWN SELF Spread
            ^avoid_attack;
        }

        if ((private->enemy_force_size) &&
            is_moving(vehicle_id))
        {
            make_umxtravel_parameters_available(vehicle_id);
            SPAWN SELF Spread
            ^avoid_attack;
        }
    }
            params

avoid_attack
    tick
    if (time_to_monitor(private))
    {
        check_enemy_force_spotted(private, vehicle_id, unit_entry);
        check_unit_under_fire(vehicle_id, private, unit_entry);

        /* FM specify long time = 60 sec. */
        if (no_air_veh_for_long_time_p(vehicle_id, private, unit_entry))
        {
            send_contact_report(vehicle_id, private);

            STOP Spread
            ^monitoring;
        }

        if ((private->under_fire) &&
            is_proper_defensive_posture(vehicle_id, private))
        {
            STOP Spread
            make_umxtravel_parameters_available(vehicle_id);
            SPAWN SELF Targeter
            ^under_air_attack;
        }
    }

        params

under_air_attack
    tick
    if (time_to_monitor(private))
    {
        check_enemy_force_spotted(private, vehicle_id, unit_entry);
        check_unit_under_fire(vehicle_id, private, unit_entry);

        /* FM specify long time = 60 sec. */
        if (no_air_veh_for_long_time_p(vehicle_id, private, unit_entry))
        {
            send_contact_report(vehicle_id, private);
            private->time_last_monitor = 0;
            STOP Targeter
            ^monitoring;
        }

    }

        params

END

SUSPEND

        /* NOP */;
```

Figure 3: AAFSM Implementation of Reaction to Air Attack Drill

In the "monitoring" state, the AAFSM performs following:

1. Continuously check whether enemy air vehicles are visible or the platoon is under air attack.

2. If the tank platoon is under-fire and it has a right defensive posture, then it has to counter attack and go to "under_air_attack" state.

3. If the platoon is moving and under fire, then make them scatter and go to "avoid_attack" state.

4. If enemy air vehicles are visible and the platoon is moving, then scatter and go to "avoid_attack" state.

In the "avoid_attack" state, the followings are executed:

[4]Design methodology for constructing a FSM can be found from [Hill 1974].

1. Update status of enemy air vehicles' visibility and enemy attacking status.

2. If more than 60 seconds after the last enemy air vehicle is spotted, then make the platoon send a contact report and return to "monitoring" state.

3. If the platoon is under enemy air vehicle attack, and they are in a right posture, then counter attack and go to"under_attack" state.

In the "under_air_attack" state, the AAFSM executes followings:

1. Update status of enemy air vehicles' visibility and enemy attacking status.

2. If more than 60 seconds after the last enemy air vehicle is spotted, then make the platoon send a contact report and return to "monitoring" state.

Surprisingly enough, the ModSAF version of React Air behavioral description seemed complete, but many details were missing. To find them, a programmer has to go through reasoning process applying what-if mental tests on all plausible cases. For example, he has to question what has to be done if no air vehicles are spotted for at least 60 seconds in the "avoid_attack" state. Obviously, the "avoid_attack" state has to be terminated and to make a state transition to the "monitoring" state. This case is not explicitly described in the ModSAF version of React Air behavior. Soon, he will realize that the "under_air_attack"state should check this condition to make a proper state transition to the "monitoring" state. This is a *default* state transition case. This type of discovery process requires a common sense reasoning or domain expertise.

The state like description in the FM or the ModSAF version React Air does not fully match with the states in AAFSM. Specifically, a portion of the "under air attack" description in the ModSAF version React Air is taken out from the "under_air_attack" state, and merged into the "avoid_attack" state because of the common observable sub-behavior; i.e., vehicle scattering. Even though this approach leads to a concise and cleaner state implementation in AAFSM, it adds an extra complication at the "monitoring" state. The "monitoring" state has to consider three state transition cases in stead of two state transition cases. The extra state transition case could have not been existed. This is an *added* state transition case to be uncovered.

5.3 RBM Implementation

RBM allows to approach the problem from a totally opposite direction; that is, rather than directly following the flow of the narrative description of the ModSAF version of Reaction to Air Drill behavior, RBM guides the behavior implementation from the goal (or objective) to simpler objectives or subgoals. In this paper, the top goal (or objective) is to react to enemy air vehicles. Then, the goal is decomposed into three subgoals, which are "under_air_attack" "avoid_attack", and "end_react_air". Though the names of the first two subgoals directly match with the two behavioral descriptions in section 5.1, the last name, "end_react_air" represents the "Report" behavioral description. The full description of the React Air behavior is shown in Figure 4.

```
react_air :- no_air_vehicle_last_60sec_p,
  end_react_air.
react_air :- under_attack_p, under_air_attack.
react_air :- air_vehicle_spotted_p, avoid_attack.

end_react_air :- was_in_reaction_p,
  report_air_vehicle_contact, react_air_done.
end_react_air.

under_air_attack :- moving_p,
under_air_attack_moving.
under_air_attack :- under_air_attack_stationary.

under_air_attack_moving :-
  proper_defensive_posture_p, counter_attack.
under_air_attach_moving :- spread_out.

under_air_attack_stationary.  ; do nothing

avoid_attack :- moving_p, spread_out.
avoid_attack.      ; do nothing

counter_attack :- helicopter_p, attack.
counter_attack :- air_veh_p, attack.
counter_attack.
```

Figure 4: RBM Prolog description of Reaction to Air Attack Drill

In Figure 4, the top goal is specified as "react_air". The "react_air" is decomposed into three ORed subgoals, "under_air_attack", and "avoid_attack", "end_react_air". Thus, if one of the subgoals is achieved, then the top goal is achieved. Therefore, a tank platoon with this "react_air" behavior implementation will always perform one of the subgoals depending on the current situation.

However, there might be a subgoal conflict among ORed subgoals; i.e., more than one subgoals might be eligible at a given condition. Specifically, more than one predicates which are attached in front of the subgoals can return "true". Then, multiple subgoals will be chosen (or selected to be executed). For example, in Figure 4, the "under_attack_p" and the "air_vehicle_spotted_p" predicate functions return true when a spotted air vehicle attacks the tank platoon.

The multiple subgoal activations generally lead to behavioral conflicts because they will make one tank platoon simultaneously perform two conflicting tasks. Thus, the subgoal conflict should be resolved. RBM uses a priority based subgoal conflict resolution scheme. If two subgoals are activated, then one subgoal having a higher priority is selected for further processing (or executed).

When some of ORed subgoals are activated for execution, the rest of them are not activated. For example, an attack from a spotted air vehicle activats the "under_air_attack" and "avoid_attack" subgoals, but it does not activate "end_react_air" subgoal. The reason is that under the situation "under_attack_p" and "air_vehicle_spotted_p" predicate functions return true, but "no_air_vehicle_last_60sec_p" returns false. In general, a given situation (or a given condition) divides ORed subgoals of a common parent goal into two groups; i.e., activated and non-activated.

In RBM, the priorities among activated ORed subgoals are statically assigned before a program execution. The priority assignment process actually becomes an ordering process of predicate condition sets whose members are conditions that satisfy the predicate function of each subgoal because the subgoal activation is directly depends on the satisfaction of the predicate function and because the predicate function's satisfaction is determined by the conditions.

Ordering of the predicate condition sets requires a proper subset relationship. If a set is a proper subset of the other set, then the former has a higher priority in RBM. Thus, the whole condition sets are partially ordered by the set priority. In the above example, there are three predicate condition sets; "under_air_attack" condition set, U, "avoid_attack" condition set, A, and "end_react_air" condition set, E. Among these sets, it can be easily observable the following relationship[5]:

$$A \supset U \quad (1)$$
$$A \cap E = \varnothing \quad (2)$$

A graphical representation of the above relationship is shown in Figure 5.

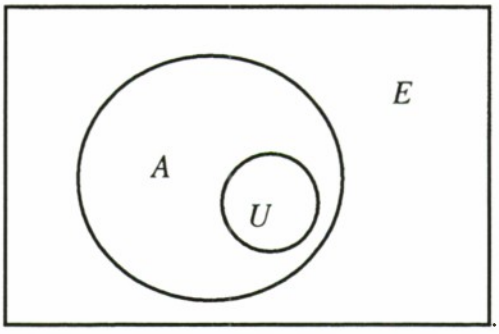


Figure 5: Relationship among Predicate Condition Sets

Consequently, there is the following set order:

Group 1:
U, A
Group 2:
E.

Though the above example does not clearly manifest a partial ordering relationship, it can be easily seen that there are at most two groups of sets at any level. That is, when following down to the partial ordering branch, there are zero, one, or two branching factors if a null set branch is allowed to be omitted. If a whole partially ordered set recursively meets this condition, then it is called *partially ordered binary set* or *partially ordered binary predicate condition set.*

When the order is found, the priorities of subgoals are automatically determined by the set order. For example, the "under_air_attack" subgoal has a higher priority than the "avoid_attack" subgoal. However, the "end_react_air" subgoal does not need to be ordered against the other two subgoals because there is no priority conflict cross the boundary of the two disjoint groups. A conflict only occurs inside a group.

There is the other type of subgoal relationship, which is an ANDed subgoal relationship. Though a similar consideration can be made, in RBM the ANDed subgoals are simply treated as a set of ordered subgoals, and tested sequentially for subgoal selection. If one of them is failed to achieve its

[5]In this problem, a direct fire air attack and a perfect vision sensor within a effective range of the direct fire attack are assumed. For example, a Hellfire (HELicopter Launched FIRE-and-forget) missile attack is not considered. This seems also true for Army FM 17-15.

subgoal, then the whole set of the ANDed subgoals is failed. Therefore, the parent goal fails.

6. Comparisons and Results

States in AAFSM are used as means to sub-divide a behavior control problem. Because of the introduction of states, a big problem becomes many smaller problems to solve. The reduced size of each sub-divided problem helps a programmer to organize his thinking; i.e., divide-and-conquer. For example, the above react air behavior was divided into three states. This state division was strongly suggested by the Army FM, and it was logically correct, too. No two states can be merged into one without behavioral crash in the merged state. However, more than three states could be used. This observation reveals that introduction of the states may not be arbitrary but constrained by the behavioral crash.

The goal decomposition in a RBM is an another means of divide-and-conquer for a behavioral control problem. It starts from a global goal and sub-divides it into subgoals. Predicate condition sets of the subgoals guide the decomposition process. Especially, OR-subgoals are created so that a partially ordered binary predicate condition set is constructed. The ANDed subgoals are simply arranged by an expected sequence of execution because a sequential execution is assumed. There is a chance that two types of subgoals coexist under a single parent goal. However, this case can be easily avoided by abstracting a series of contiguous ANDed subgoals into a single subgoal so that all subgoals under a single goal become either ANDed subgoals or ORed subgoals. Practically speaking, the mixture of two types is allowed as long as a series of ANDed subgoals is effectively treated as one subgoal.

Both approaches have a divide-and-conquer mechanism; i.e., means of problem decomposition scheme. However, their directions of approach are totally opposite. AAFSM starts from a known start state and moves toward the goal state, while RBM starts from the goal and decomposes it into subgoals. The state decomposition of AAFSM (or FSM in general) is constrained by behavioral crash (sort of output side), but the goal decomposition of RBM is guided by predicate condition sets of subgoals (sort of input side). This observation reflects the opposite reasoning philosophies. In order to move forward to a goal, the output of the current (or future) state is important, but to move backward from a goal toward a given initial condition, the input condition check is important in order to know the arrival of the initial condition.

Not all theoretically possible state transitions are actually implemented in an AAFSM, but all the possible transitions have to be considered at least once to determine the possibility of inclusion to the AAFSM. For example, in the reaction to Air Attack Drill behavior, there were 3 states. Thus, total 6 state transitions had to be considered. Among the six, one transition turns out to be not needed because the state transition from the "under_air_attack" state to the "avoid_attack" state is not feasible. If an enemy aircraft spots a tank and starts to attack, there is no reason that the enemy simply stops attacking[6].

The state creation is not an easy task in a big sized problem, but complete consideration of all possible state transitions is even harder. For example, if there are four states in an AAFSM, then total 12 different state transitions have to be considered. If 5, then 20 possibilities, and so on. In general, if there are n states, then (n-1)*n transition conditions have to be considered. Obviously, this is a case of combinatorial explosion.

All of the theoretically possible state transition cases will not be actually included in an AAFSM, and the ratio between the number of practically feasible state transitions and the total possible state transitions rapidly decreases as the number of the states increases. However, at least consideration of all possible cases should be done.

The numerous state transition cases of AAFSM (or FSM) tend to cause a programmer failing to include all necessary state transition cases. The omission becomes a common cause of bugs for behavioral implementations. Even though a theoretical number of transitions from a state is known, this is a very weak checking mechanism. The added and default state transition conditions make a systematic checking of the omission even more difficult.

In a RBM design/implementation, a goal is recursively decomposed by the predicate conditions of subgoals. If the subgoal decomposition produces a partially ordered binary set, then the priorities of subgoals are already determined. The goal having the highest priority is placed at the first place, and then the goal with the second priority is the next, and so on following the partial order. Therefore, a RBM design/implementation becomes searching for a

[6]However, in the actual implementation, 6 state transitions were included in the AAFSM because of the added case caused by the AAFSM state mismatch to the ModSAF (or Army FM) descriptions.

subgoal decomposition that produces a partially ordered binary set.

If proper subgoals with a partially ordered binary predicate condition set are found, then there is no need to consider another step as the AAFSM based design, because all possible state transition conditions are implicitly included in the subgoal priorities. For example in the RBM Prolog implementation in Figure 4, three lines of the Prolog code with the "react_air" header automatically enumerate the 6 possible state transition cases considered in the AAFSM implementation. Again this is not a surprising result. RBM subdivides a problem by optimizing predicate conditions of subgoals (or roughly similar to the state transition conditions). The omission problem of AAFSM is systematically prevented in RBM. For example, three lines of the code automatically took care of the 6 possible transition cases.

The experience in implementing the React Air behavior in AAFSM and RBM shows that RBM implementation is more natural than the other. The military system is one of the most highly optimized man-made systems. It has a really long history (experience) and has gone through numerous selection processes; wars. Thus, most of the knowledge about their behaviors, called military doctrines and tactics, should be well developed and organized. For example, the Reaction to Air Attack Drill behavior of Army FM is effectively written in a backward reasoning representation even though it gives an impression of a time-line based action description. It basically expresses what is the global objective and what are the sub-objectives and associate actions, and so on. It even tends to implicitly prioritize sub-objectives too. It describes the "Seek cover and concealment" sub-objective in following: "When moving, tank units must immediately seek cover and concealment." Cover and concealment is the best behavior to achieve defensive reaction to air attack. This description is immediately followed by "If concealment is not available, moving tanks should stop." which is the second best to avoid air attack.

7. Future Works and Conclusions

There exist many behavioral representations other than AAFSM(or FSM) and RBM; such as Petri Nets, Production Rule based systems, and subsumption architecture, etc. Because almost all of them are fundamentally classified as a forward reasoning representation, it is expected that they will have similar characteristics of AAFSM (or FSM). However, they deserve careful investigations because of their additional characteristics over AAFSM (or FSM).

A Petri Net is a super-set of a FSM (Murata 1989) because of its equivalent expression power of Turing machine. It might provide a better behavioral representation than a conventional FSM. A production rule based system allows to incrementally add each fragment of behavioral knowledge in IF-THEN rule forms. Because of this attractive nature, there are quite number of implementations (Scholz 1993, Byrnes 1992, Giarratano 1991, Rosenbloom 1993). However, this modularity can easily lead to unexpected interactions to the existing rules in the system when more and more rules are accumulated in the system. Finally, the subsumption architecture (Brooks 1986) including later variations might give a better behavioral representation. Especially, modified subsumption architecture tends to eliminate a pure optimistic behavioral implementation. However, no significant fundamental departure from the original data-driven forward reasoning has been observed.

Behavioral complexity of current CGF seems not yet to reach the point where it clearly manifests characteristics and limitations of each behavioral representation approach. However, the future CGF's behaviors will be much more complex, and the scope of CGF broadens. Then a behavioral representation will be getting more important. A better representation with a closer domain match to a target behavior description, such as FM or knowledge from a subject matter expert, is expected to be a winner by saving development and maintenance time and efforts. This comparison study shows that a backward reasoning representation has an advantage over the other representation because of the close domain match with the military's behavioral knowledge representation.

8. Acknowledgments

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001-0058.

9. References

Army (1987). *Tank Platoon*, FM17-15, US Army, October.

Brooks, R (1986). "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol, RA-2, No.1., pp. 14-23.

Byrnes, R.B., McPherson, D.L., Kwak, S.H., McGhee, R.B., and Nelson, M.L. (1992). "An

Experimental Comparison of Hierarchical and Subsumption Software Architecture for Control of an Autonomous Underwater Vehicle", *Proceedings of 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology*, Washington D.C., June 2-3, pp.235-241.

Byrnes, R.B. (1993). *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles*, PhD Dissertation, Naval Postgraduate School, Monterey, CA.

Hill, F.J., Peterson, G.R. (1974). *Introduction to Switching Theory and Logical Design*, 2nd ed., John Wiley & Sons, New York, New York.

Homem de Mello, L.S., and Sanderson, A.C. (1991). "Representations of Mechanical Assembly Sequences", *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 2, April, pp. 221-227.

Jackson, P. (1990). *Introduction to Expert Systems*, 2nd Ed., Addison-Wesley, Reading, MA.

Kwak, S.H., McGhee, R.B. (1990). "Rule-based Motion Coordination for Hexpod Walking Machine", *Advanced Robotics*, Vol. 4, No. 3, 1990, pp. 263-282.

Kwak, S.H., McGhee, R.B., and Bihari, T.E. (1992). *Rational Behavior Model: A Tri-Level Multiple Paradigm Architecture for Robot Vehicle Control Software*, Technical Report NPS CS-92-003, Naval Postgraduate School, Monterey, CA.

Kwak, S.H., Scholz, T., and Byrnes, R.B. (1993). *The State Transition Diagram with Path Priority and Its Applicability to the Translation between Backward and Forward Implementations of the Rational Behavior Model*, Technical Report, CS-93-003, Naval Postgraduate School, Monterey, CA.

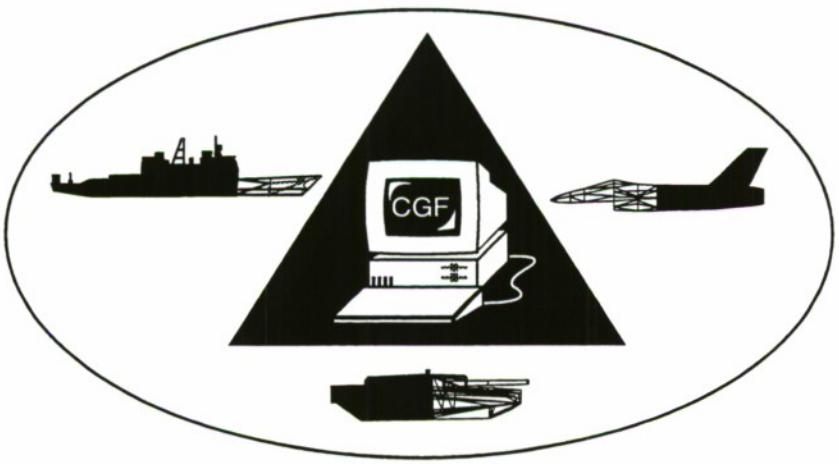
Loral (1995). *ModSAF Software Architecture Design and Overview Document*, Loral, Cambridge, MA.

Rosenbloom, P.S., Laird, J.E., Newell, A., and McCarl, R. (1993). "A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence", *The Soar Papers*, Vol. 2., The MIT Press, Cambridge, MA, pp.860-896.

Scholz, T. (1993). *The State Transition Diagram with Path Priority and Its Applications*, MS Thesis, Naval Postgraduate School, CA.

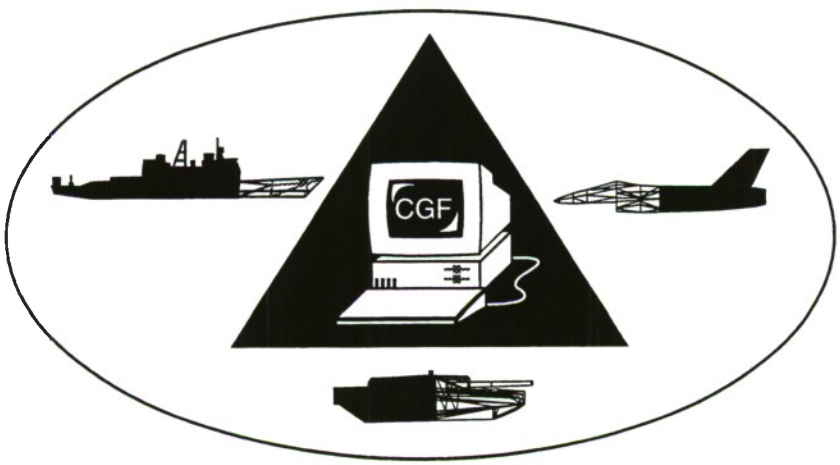
10. Biography

Se-Hung Kwak is Principal Software Engineer at Loral, Advanced Distributed Simulation, where he has led development and application of software architecture for Computer Generated Forces since 1993. He received his Ph.D. and M.S. from Ohio State University in Electrical Engineering in 1986 and 1984 respectively, and his B.S. from Seoul National University in Electronics Engineering in 1979. Before joining Loral, he was Research Associate Professor of Computer Science at Naval Postgraduate School. His research has focused on intelligent robot control and control architecture. He is also the originator of the Rational Behavior Model(RBM). His research interests include Artificial Intelligence, Robotics, Mission Planning Expert Systems, Object-Oriented Parallel Systems, Software Engineering, and GPS/INS Navigation Systems.


CGF

Session 10b: General Interest

Mellies, TRADOC DCSINT
Pickett, TRADOC Analysis Center
Williams, University of Virginia


CGF

The OPFOR Model in CCTT and Beyond: Applications in DIS

Penny L. Mellies
TRADOC DCSINT, Threat Support Division
Ft. Leavenworth, Ks
melliesp@leav-emh1.army.mil

1. Abstract

In the future, all people, forces and institutions in the defense community will use interoperable simulators, simulations, and fielded systems that simultaneously interact on a shared synthetic battlefield that realistically represents warfighting concepts, doctrine, forces and weapon systems of friendly, neutral and opposing forces. (The DIS Master Plan, 1994)

The Distributed Interactive Simulation (DIS) Master Plan calls for a flexible, consistent, validated, intelligent and doctrinally-based computer generated opposing force (OPFOR). In its mission to serve as the TRADOC ODCSINT point of contact for all intelligence, threat and opposing force (ITO) initiatives to DIS, the TRADOC Office of the Deputy Chief of Staff for Intelligence (ODCSINT)Threat Support Division (TSD) has produced two OPFOR models for use in DIS-compatible simulations and simulators.

These unclassified models provide flexibility and a *capabilities-based* OPFOR which can be tailored to represent a wide-range of potential capabilities and organizations. The Heavy and Light OPFOR packages provide the doctrine, tactics, and equipment data needed to develop OPFOR behavioral representations for a semi-automated opposing force (SAF OPFOR). The OPFOR model captures tactical and operational representative behaviors and provides the organizational building-blocks necessary to operationalize a SAF OPFOR.

The DIS environment requires the development of a model that provides a traceable, consistent, flexible and doctrinally-based OPFOR. TSD was challenged to develop a new OPFOR that provides increased flexibility to accommodate a wider range of training and simulation requirements. These training requirements will span the entire operational continuum, from heavy mechanized forces to operations other than war (OOTW) and post conflict operations. In addition to the expanded range of operations, the capabilities-based model is required to consider the worldwide proliferation of weapons and their various tactical applications.

This paper will present the TSD OPFOR model and its use in the development of DIS-compatible simulations. It will, in essence, present a progress report on the development of the model and its application in the DIS environment.

Currently, the TSD-produced Heavy OPFOR is being used in the development of the Close Combat Tactical Trainer (CCTT) and Warfighter's Simulation (WARSIM) 2000 OPFOR. CCTT continues to evolve as the prototype for ITO simulation support, so the need to understand the TSD OPFOR model and its current and potential applications becomes even more critical.

The new OPFOR model is capabilities-based, not country-based. The model provides a tailorable force structure in the form of a Heavy and Light OPFOR package. The model accommodates existing models while allowing for evolutionary change and user feedback. The outcome of the final OPFOR product is a validated, flexible, consistent, and doctrinally-based paradigm for the creation of a SAF OPFOR.

This paper will explore the methodology used to construct the capabilities-based OPFOR, and its value in DIS-compatible simulations across all three DIS domains (TEMO, RDA, ACR).

2. Introduction

CCTT is the Army's number one priority training device. Using a system of combat vehicle simulators and computer workstations, armor, cavalry and mechanized infantry units will train to battalion/task force level.

CCTT is the first in a series of five simulations belonging to the Combined Arms Tactical

Trainer(CATT)family. CCTT serves as the prototype upon which future simulations will be built.

A realistic OPFOR is necessary in any successful training environment. CCTT is no exception. This semi-automated OPFOR must be doctrinally-based, flexible and adaptable to all levels of conflict. OPFOR accountability and traceability are paramount to the success of CCTT and its follow-on programs. The Army cannot afford to spend scarce resources and limited manpower on the duplication of data collection and application efforts.

The Army mandated that an accountability and verification methodology be established for all data and behavioral capabilities used to develop the OPFOR. TSD's OPFOR model provides this methodology.

Thus, TSD was tasked to develop an OPFOR model that would support Force XXI training and support needs and simulation requirements in the DIS environment.

Future applications of the model will be developed for the Advanced Concepts and Requirements (ACR) and the Research, Development and Acquisition (RDA) domains. TSD's goal is to provide ITO support in adequate fidelity and detail to facilitate realistic simulation across all three DIS domains : ACR, RDA and Training, Exercise, and Military Operations (TEMO).

3. Background

In 1993, the US Army's Simulation, Training, and Instrumentation Command (STRICOM) and Integrated Development Team (IDT) began the Close Combat Tactical Trainer (CCTT) SAF Functional Analysis to "define the scope of tactical behaviors" for both the BLUFOR and the OPFOR. STRICOM and IDT agreed to use Army Training and Evaluation Programs (ARTEPs) as the foundation to define BLUFOR tactical behaviors.

Such a well documented body of literature did not exist for the OPFOR. TSD, STRICOM and IDT worked in concert to produce an initial list of OPFOR behaviors to be represented in Combat Instruction Sets (CISs). The result was more than 500 OPFOR tasks and close to 40 OPFOR units. (For a detailed description of the CISs and the validation process, see McEnany and Marshall. "CCTT SAF Functional Analysis", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, 1994, page 195).

CISs were written to insure that the proper tactical behaviors were translated into software code. Each CIS provides a detailed explanation via text, sketches, and movement parameters of all actions necessary to complete a task.

The CIS process provides the accountability of OPFOR sources and overall task development, called for in the Army mandate.

Initial CISs are developed by SAIC (Denver) and passed through TSM-CATT to TSD for review and approval. The Heavy OPFOR package is the baseline for all the CCTT CIS development. All OPFOR CISs must be validated by TSD before being operationalized in CCTT.

In addition to the validation of the CISs, other TSD responsibilities for CCTT include: the collection and accountability of data used in the simulation; providing the interface between the intelligence community and the developers to answer all requests for information (RFIs); participation in the Subject Matter Expert (SME) Network; and providing OPFOR representation in Working Group and User Exercises. Each of these responsibilities allows TSD to ensure the accurate and consistent application of the OPFOR model in CCTT and future DIS-compliant simulations.

4. Catalysts for Change

With the collapse of the monolithic Soviet threat, an OPFOR based on a single threat country was no longer feasible or realistic. The traditional country-based OPFOR was now antiquated and incapable of capturing the full spectrum of potential military operations.

The Cold War paradigm for threat/OPFOR analysis was in need of repair. The Threat Spectrum Model (TSM)(Figure 1) was developed to present potential threats across "a spectrum from simple to complex in scope, doctrine, organization, training, materiel, leadership and soldiers."(TRADOC PAM 525-5). Using the TSM as a springboard, TSD began to develop the new OPFOR model.

The model captures the capabilities of a vast range of potential OPFORs and provides a flexible training force that can be tailored to represent this variety of potential threat capabilities, organizations and equipment.

Creating a model that was based on capabilities, rather than a specific country's military potential, provides a more flexible and dynamic training tool for the BLUFOR, and offers OPFOR commanders/operators a varied range of tactical options.

The capabilities-based model has become the basis for the forces and doctrine used by the OPFOR at the Combat Training Centers (CTCs), and has been integrated into the latest TRADOC Common Teaching Scenario at the Centers and Schools.

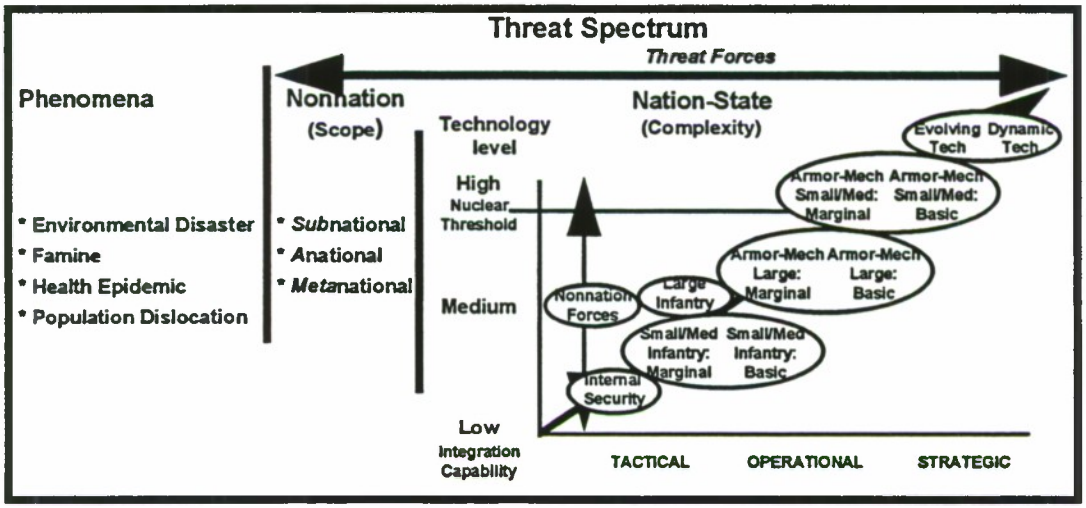


Figure 1: Threat Spectrum Model(TRADOC PAM 525-5)

5. The Capabilities-based OPFOR Model

The OPFOR model is based on a consistent and traceable body of military doctrine and tactics, ranging from the heavy end (Soviet-style) to the light (Third world and irregular) forces, as represented in the Heavy and Light OPFOR Handbook series. The six handbook package provides: an organization guide, an operations handbook, and a tactics handbook for both heavy and light forces. The equipment guide and OOTW handbook will follow the completion of the Light package in March 1995.

The success of the model is evidenced by its selection and funding for publication as Army Field Manuals (FMs). The transition to FMs will occur in Fall 1995.

This new series of handbooks was originally developed by TSD to support Force XXI training and development needs. However, by the completion of the organization guides, the model had been embraced by the simulation community for use in CCTT. This was a natural evolution of the model's application. It made perfect sense for the model being used for live training and instruction to also be applied in the virtual and constructive simulation arenas. OPFOR linkage and continuity was established between arenas. Thus, the model became the cornerstone for all ITO training and instruction at the CTCs, TRADOC Schools and Centers and the development of the SAF OPFOR for CATT and FAMSIM programs. (Figure 2)

The Heavy and Light packages are based on a mixture of doctrine, organization and tactics of foreign armies. However, the OPFOR packages are not simply a collection of unclassified handbooks explaining how a particular foreign army fights. They are composites constructed to provide the customer a wide-range of OPFOR capabilities. The packages do not mandate a fixed order of battle, but rather provide the building-blocks from a variety of potential orders of battle. (Figure 3)

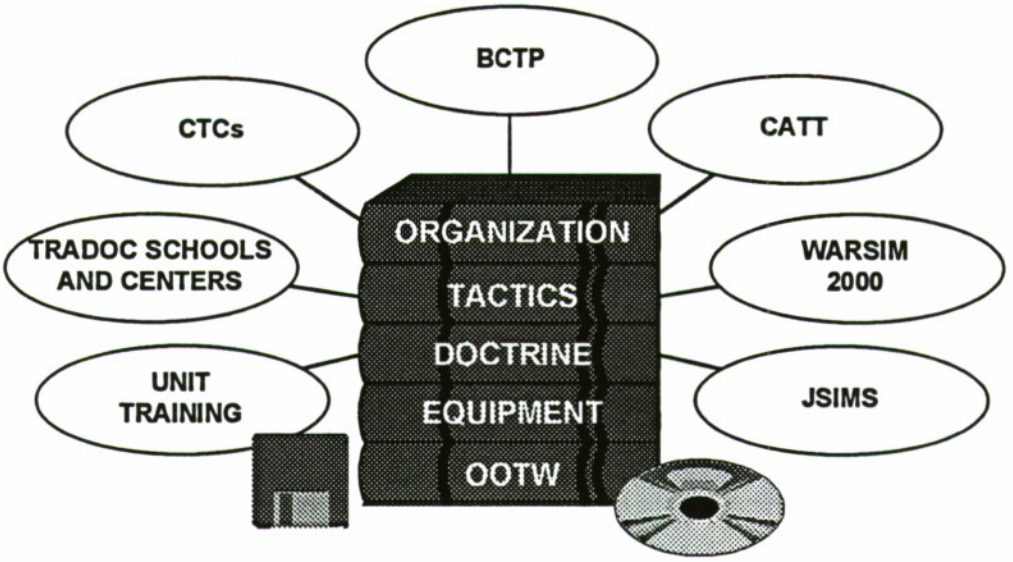


Figure 2: Consistent OPFOR Model Application

OPFOR MODEL EXECUTION

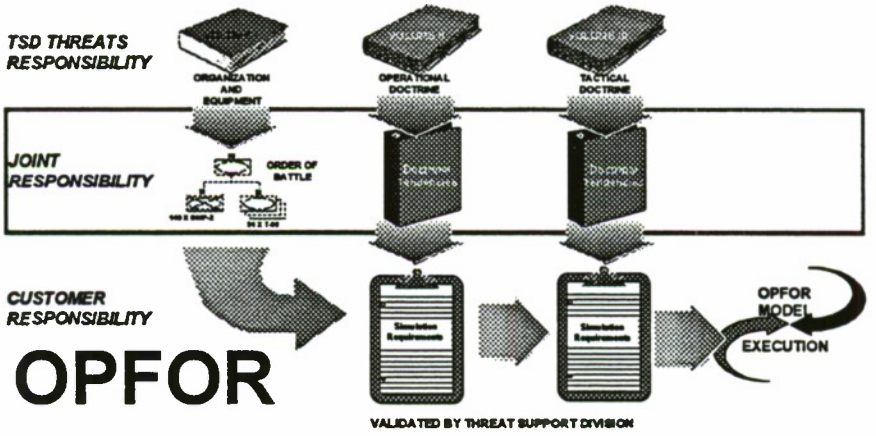


Figure 3: OPFOR Model Execution

5.1.The Heavy OPFOR Package

The Heavy OPFOR was initially based on a Soviet-type military force. As the model evolves, more options will be included to offer increased flexibility. The Heavy OPFOR structure consists of regiments, divisions, and armies, as well as brigade/corps and an infantry division. The organization guide's "building-block" approach allows the customer to select

the needed organizations and structure based on his requirements. This approach allows for a wide-range of options to build various Heavy forces. The equipment includes Former Soviet Union (FSU) combat systems as the baseline. However, a worldwide equipment substitution matrix is included to allow the user to substitute pieces of equipment to fit any scenario.

5.2.The Light OPFOR Package

The Light OPFOR is based on a variety and mixture of Third World-style forces. Unlike the Heavy OPFOR, little documentation existed upon which to build the Light forces. Extrapolation of existing light forces' doctrine and capabilities was necessary.

The Light OPFOR presents the forces of a country divided into military regions with subordinate military districts. The force structure includes brigades, divisions, militia and commando forces. Most of the forces are located within the districts and vary in strength and overall capabilities.

Light OPFOR units could potentially range from a single light infantry brigade to a mechanized infantry division. The model provides the information necessary to build standing armies or divisions; however, separate brigades will probably form most of the light forces.

6. Future Application

Through CCTT, the OPFOR model was established as the prototype for the SAF OPFOR in the CATT family of simulations. This ongoing OPFOR development process is critical to the success of CCTT and will play an integral part in future CATT Programs and several Family of Simulators (FAMSIM) projects, i.e., WARSIM 2000 and JSIMS. The consistent, evolutionary application of the model also insures that the individual soldier will be able to "follow" the model through his training career. TSD continues to evolve and expand the OPFOR model and respond to the simulation community's ITO needs. The need for a validated, consistent, flexible and doctrinally based SAF OPFOR has clearly been established. TSD's OPFOR model satisfies these requirements while retaining a consistent, traceable doctrinal base.

WARSIM 2000 is the Army's next-generation simulation for supporting Force XXI Battle Command training. The objectives of WARSIM can successfully be obtained through the application of the OPFOR model. For example, the model will support Army and Joint Force Training in scenarios across the operational continuum, and will provide a flexible base for growth while reducing the resources required to support training.

7. Conclusions

Distributed Interactive Simulation (DIS) is the future of military training, testing, research and development. Computer Generated Forces (CGF) are a critical component of the DIS environment and technology. The success of the simulation depends on an accurate and consistent application of a computer generated OPFOR. CGFs provide the most economically feasible method for populating the synthetic battlefield.

The TSD-produced OPFOR model provides the necessary doctrinal and tactical support to develop such forces, specifically the SAF OPFOR for CCTT and WARSIM 2000. The model provides the much needed documentation to develop OPFOR tactical behavioral representations in DIS.

The unclassified model also provides an 80% solution for ACR and RDA OPFOR needs. However, the requirements for classified ITO support will be developed by the various national agencies. Current intelligence and specific country capabilities will supplement the OPFOR model. TSD will support these requirements as needed. The Threat Support Division will continue to support all three DIS domains with selected OPFOR data and validation. DIS will continue to evolve as a critical tool for military training and development. Thus, the application of TSD's OPFOR across various simulations and in multiple training environments, provides consistency and accountability in the DIS environment.

8. References

CAC Threats. (1994) "The Capability-Based OPFOR," *Threats Update*. 63 pages.

CAC Threats. (1994) "Threat Force Package," *Threats Update*. 63 pages

Ceranowicz, A. (1994). "ModSAF Capabilities," *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation. STRICOM, DMSO and IST.* 544 pages.

HDQA, Office of the Assistant Deputy Chief of Staff for Operations and Plans. (1994). *Army Modernization Plan Distributed Interactive Simulation (DIS).*

The DIS Master Plan. (1994).

McEnany, B.R., Marshall, H. (1994). "CCTT SAF Functional Analysis," *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*. STRICOM, DMSO and IST. 544.

Memorandum For TSD from DAMI-FIT, (1995) Subject: Minutes of Distributed Interactive Simulation Threat Support Working Group (DIS-TSWG), 7-8 Feb. 95. 6 pages.

Memorandum For TSD from TSM-CATT, (1994). Subject: CATT Fact Sheet. 4 pages.

National Simulations Center. (1995) "The Functional Description of the Battlespace", 22 pages.

National Simulations Center. (1995) "Warfighter's Simulation 2000 (WARSIM) Overview," 39 pages.

TRADOC DCSINT, TSD Mission Briefing, February 1994.

TRADOC PAM 525-5 (1994) *Force XXI Operations: A Concept for the Evolution of Full-Dimensional Operational for the Strategic Army of the Early Twenty-First Century.*

Wright, R. (1994) "Source Data Acquisition For the Close Combat Tactical Training Systems and Education Conference," *Proceedings of the 16th I Interservice/Industry Training Systems and Education Conference.*

9. Authors' Biography

Penny L. Mellies is an Intelligence Research Specialist at the Threat Support Division, TRADOC DCSINT, Ft. Leavenworth. Ms. Mellies is responsible for ITO support to CATT and FAMSIM projects. Previously, she served as co-author for the OPFOR Heavy Package. She has a Masters of Arts degree in Political Science and International Relations from the University of Kansas. She is a 1992 graduate of the Army Intelligence Internship Program and the Military Intelligence Officer's Basic Course (MIOBC). Her research interests include ITO support to simulations and OPFOR development and applications in DIS.

Report on The State of Computer Generated Forces 1994

H. Kent Pickett
TRADOC Analysis Center
Ft. Leavenworth KS 66027

Mikel D. Petty
Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826-0544

1. Abstract

Computer Generated Forces (CGF) systems in Distributed Interactive Simulation (DIS) are almost always designed, developed, and described with training applications in mind. This paper surveys the state of DIS CGF development from a broader point of view, examining the capabilities of existing CGF systems relative to applications in training, advanced technical demonstrations, and analysis. Those capabilities are evaluated against the requirements stated in the U. S. Army's Distributed Interactive Simulation Master plan. The paper concludes with a list of problem areas and recommendations for producing standardized CGF systems.

2. Introduction

In May 1994 the Army Modeling and Simulation Master Plan was published. This plan names 20 areas for consideration in the Army Modeling and Simulation Community. Within each area, software standards are to be established and technical procedures are to be defined for implementing these standards across the modeling community. The Master Plan further encouraged standards for development through the establishment of "teaming arrangements" and "consensus building" within the Army modeling community. TRAC was given the responsibility for the area of Computer Generated Forces (CGF). This report represents the initial effort by the coordinator in CGF (the first author). Its goal is to set the tone for developing a standard CGF software system with emphasis on FY95-FY96.

The report is divided into three sections. It begins with a section describing possible uses for CGF. In addition to the current use with the Army training community, uses in combat developments and force structuring are also described. The section concludes with a set of goals for CGF systems reflecting the suggested uses. The report continues with a short tutorial on the development of a CGF software system. The tutorial includes a description of the principal software structures found in every CGF architecture. It is within the context of these structures that specific areas requiring standardization are discussed. The section also contains a description of the current "state of the art" in each area. The final section of the report summarizes those areas in CGF development requiring attention in terms of research and development resources. This section is provided as a guideline for suggested resource allocation.

Two source documents were very important to this report. The first is the *1993 DMSO Survey of Semi-Automated Forces* (Booker,1993). The survey was sponsored by DMSO and conducted by a team from MITRE and IDA. The document surveyed eight projects developing CGF systems and described the architectural structure of each. It also reviewed six efforts conducting research related to CGF. Many of the thoughts in this report describing problems with current CGF systems were taken from the DMSO survey.

The second source document is the U. S. Army's *Distributed Interactive Simulation (DIS) Master Plan* (U.S. Army, 1994). It describes the Army's requirements for DIS and a strategy for investments and research priorities with the goal of producing a useful DIS system. The recommendations for AMIP/DMSO support in CGF research contained in the last section of this paper reflect the priorities in the DIS Master Plan.

3. Uses of/Goals for Computer Generated Forces

CGF Systems had their origin in the SIMNET environment in the mid 1980s. Their initial use was to provide a set of threat vehicles (and in some cases, to augment the live friendly forces) to train personnel in the SIMNET simulators. The two main current

CGF systems, ModSAF and the CCTTSAF, remain focused on personnel training.

However, since 1990 CGF systems have been used in other ways. In particular, they have been used for providing both friendly and threat forces and a virtual experimentation environment. This has been the case with many of the WARBREAKER efforts where CGF systems were used to provide target arrays (of both aircraft and vehicles) and the virtual battlefield (to include terrain and battle environment). These CGF-based virtual battles have been interfaced with sensor simulators (J-STARS) and live personnel in the command and control (C2) network making decisions on interdiction of deep targets. One can claim training of the C2 personnel but most often the CGF systems are the virtual backdrop against which a new attack capability is demonstrated and analyzed. This use of CGF systems is expected to continue with more complex (larger forces of both live and virtual systems) Advanced Warfighting Demonstrations (AWDs) and Advanced Technology Demonstrations (ATDs). In the case of AWDs, the principal use will be to look at new force structures and new doctrinal concepts (i.e., using groups of recon helicopters to call in deep strike artillery) and to develop Tactics, Techniques and Procedures (TTPs) with new equipment. An excellent example use of a CGF system for an ATD is the Anti-Armor Advanced Technology Demonstration; in the first A2ATD experiment, ModSAF was validated against two M1A2 Initial Operational Test and Evaluation battles (Thomas,1995).

The analytic community has also begun to consider the use of virtual based CGF systems. While the community has long used constructive models in the analysis of future systems and force structures, in the 90s much analysis will be built around a constructive+virtual concept. In this context, a portion of a lengthy battle is run in a constructive simulation in search of battle vignettes which will expose key strengths and weaknesses of the system, doctrine or force structure being analyzed. The vignette is then represented by a virtual CGF environment normally with man-in-the-loop participation. The constructive model may be suspended during this virtual exercise or may be run at real time to provide the battle context of supporting units surrounding the virtual battle. During the CGF exercise, key parameters about how humans will actually perform with the new system or tactical concept will be collected. These parameters can then be used to extrapolate the battle in the constructive model to a broader context. (See (Franceschini,1995) for a tutorial on how constructive and virtual simulation systems are linked, and (Kraus,1995) for a survey of such linkages.)

The following list is a minimal set of objectives which must be met by any CGF system which will be accepted as a "standard".

- The CGF system must be useful to all three applications (training, advanced technology demonstrations and analysis). This goal implies some specific requirements. These include:

  Ability for man-in-the-loop simulators to interface at any echelon.

  Ability to interface with live systems (actual equipment vs simulator) at any echelon.

  Ability to run real-time and, for analytic applications, faster than real-time.

  Ability to interface with constructive models in the constructive+virtual environment.

- The CGF system must be DIS compatible. The system must also have the ability to operate across both local and wide area nets. This implies that man-in-the-loop simulators may be at geographically remote locations. Further, actual portions of the CGF system (the simulation processors or the operator interfaces) may be located at geographically remote processors.

- The CGF system must represent the battle from Corps to individual vehicle.

- The CGF system must interface with other Service models in a Joint exercise.

- The number of operators in the CGF system must be minimal. It is recognized that when the system is used in training or advanced technical/warfighting demonstrations, the virtual battle must interact realistically with the exercise players. The current state of the CGF art requires some operator control of high priority vehicles and critical units in executing a dynamic and reactive virtual battle.

- The structures and data bases simulating the physical and cognitive/tactical behaviors of the vehicles, personnel and units must be modular and easily isolated for Verification, Validation & Accreditation (VV&A). The term "algorithms" has been purposely avoided in this goal since many CGF systems use finite state machines (FSMs) to represent entity behavior. In this case, the structure of the state transition tables and the time step of the FSM system are important elements to be considered in the VV&A process.

The list of goals in the previous paragraph are not all inclusive. They are only a first cut at what will be "necessary capabilities" if the CGF is to have general use among the training, analytic and technical experimentation/demonstration communities. For many of the goals, technical feasibility has already been demonstrated. However, a sufficiently robust CGF software system capable of use in a general production environment has yet to be developed.

4. Key Software Components of a CGF System

The architectural development of any CGF system demands the design of certain components. These components provide the simulation with the ability to represent the physical environment in which the battle will take place, to represent the combatants themselves (vehicles, aircraft, personnel, ships, etc.) and to represent a C2 structure for organizing the individual combatants into units and a fighting force. The CGF system further demands the development of services supporting a distributed simulation. These include network functions for putting packets on the net to update the state of the simulation and system administration functions (sometimes called controller or simulation management functions) for starting the simulation, i.e. getting all the processors connected to the network and in time sinc. Figure 1, taken from (Booker, 1993) lists these components (simulator, player, controller and management functions) and their subfunctions. For a better understanding of the functions of each component consider a CGF producing a virtual tank moving across the terrain in search of a firing position. The simulation of the terrain, tank movement rate, and the cognitive determination by the crew of a location for a firing position is done by the environmental, physical and C2 process models shown under Simulator Functions in Figure 1. As the virtual tank moves, the CGF operator watches it on his 3-D display and may send a message, using the C2 support tools to the virtual tank to "adjust" the final overwatch position. Further, as the virtual tank moves, Protocol Data Units (PDUs), messages describing tank position and appearance, are broadcast on the DIS network so that simulators in the immediate area can see his movement. The point of this example is that there are several areas that must be considered when developing a "standard" CGF system.

Before leaving Figure 1, one other point about CGF architecture should be made. Most CGF systems are truly distributed simulations running on several processors. The shaded functions in Figure 1 can be thought of as individual computers netted together forming the CGF system. For example, a typical CGF system run on five processors. Two of the processors are used for CGF operator functions (one for the operator controlling Red forces and another for Blue). Two are used to simulate the environment and the vehicles in the battle (Simulator Functions) and the fifth is used for the Distributed Simulation Management functions. See (Petty, 1992) or (Petty, 1995) for more detailed discussions of CGF system architecture.

The Army's DIS Master Plan contains a "road map" for the development of Automated Forces (aka CGF). This road map is shown in Figure 2. As would be expected from any potential user, the plan is focused on desired functionality. However, upon careful inspection of Figure 2, the focus in the early years (1994-1995) is on the development of a physical environment allowing virtual vehicle commanders to perform METT-T (Mission Enemy Troops Terrain and Time) activities and standard algorithms for performing the target engagement process. When comparing Figures 1 and 2 one can easily relate the functionality desired in the DIS Master Plan with the CGF architectural features described by the DMSO report. The desired virtual environment for METT-T in Figure 2 corresponds with the Simulator Functions in Figure 1. A second area of focus in the Automated Forces road map is the development of a structure for command and control of virtual forces. Note the block for "tactically and doctrinally correct execution of orders" leading to the "development of C2 of entities". This area again corresponds to the Simulator Functions in Figure 1. One other area of the road map contains references to "communications software" , "build PDUs. . .", and "modify. . .to run multiple processors. . ." which can be related to

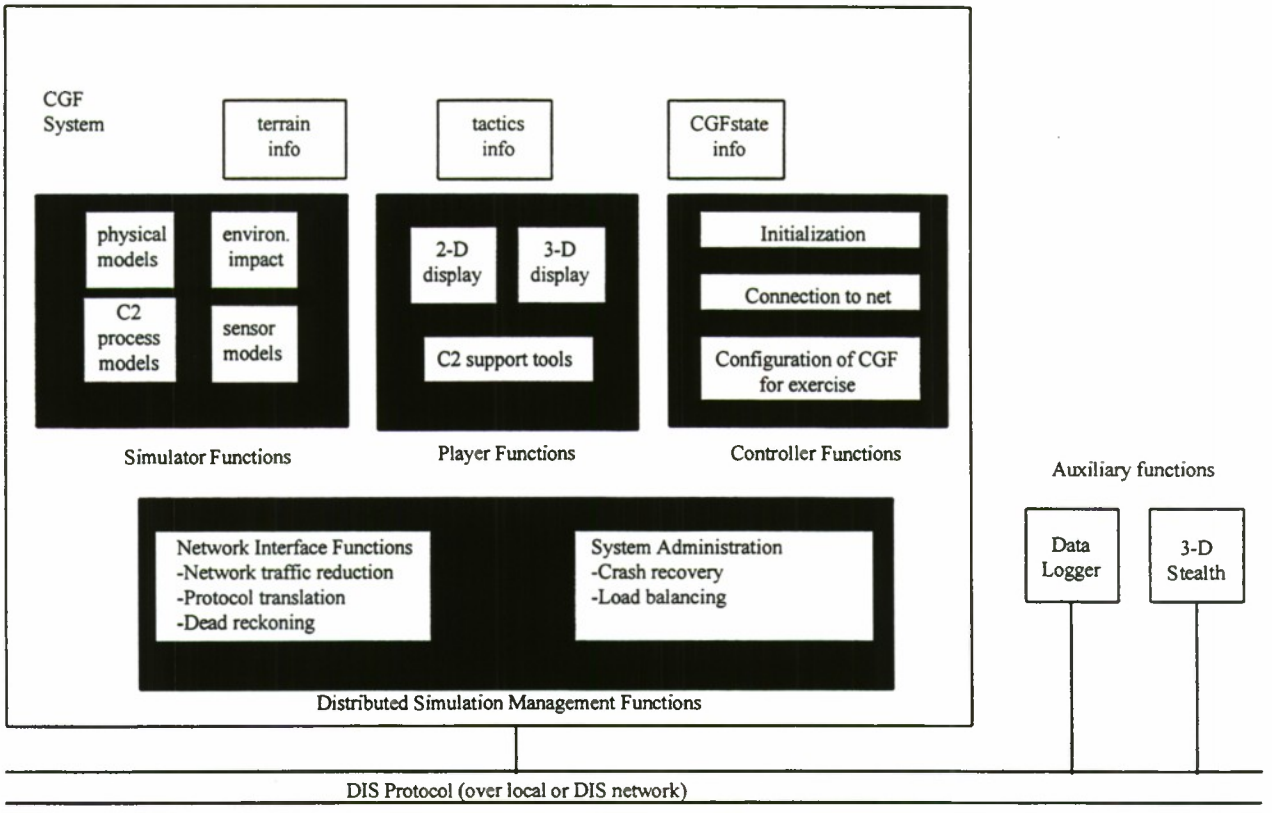


Figure 1- Notional view of the objective SAFOR system.

Distributed Simulation Management Functions in the Figure 1 architecture.

The Master Plan provides a view of where the Army wishes to focus its efforts and the time frames of particular accomplishments. Like most plans, it is optimistic. The first author believes that many of the CGF improvements (specifically dynamic terrain, weather and atmospheric signature transmission) "enabling METT-T" will not be complete until 1996-1997. However, the plan must be used as a guideline for establishing priorities in CGF improvements and standards. In that spirit, the following areas should be the focus of AMIP support for FY95-FY96.

4.1 Physical Environment

The environmental component for the Army operations consists of a representation of the terrain and low atmosphere (5K-10K feet). The objective for a CGF simulated environment must be the ability of the individual vehicular commander (mounted, dismounted or pilot) to perform METT-T activities. The CGF must provide a basic representation of environmental signature transmissions (at a minimum in the visual, thermal and radar spectrums) and effects of weather, battlefield haze and smoke on these transmissions. Further, the effects of dynamic changes in terrain (in particular, modification of terrain by battle activities) must be represented. While basic research is ongoing in these areas (e.g. for dynamic terrain see (Moshell, 1994) and (Kilby, 1994)) their conditions are listed in the DIS Master Plan as environment:red, and terrain:amber. The criticality of the physical environment and terrain in military activities makes these efforts a high priority for AMIP support.

4.2 Entity Representation

The physical representation of vehicular platforms (i.e. vehicle dynamics) to a degree of fidelity sufficient at least for training has been well established in various CGF systems since SIMNET. More recently, ModSAF's modular structure makes it possible to easily assemble physical models of vehicles from a library of existing software routines (Ceranowicz,1994).

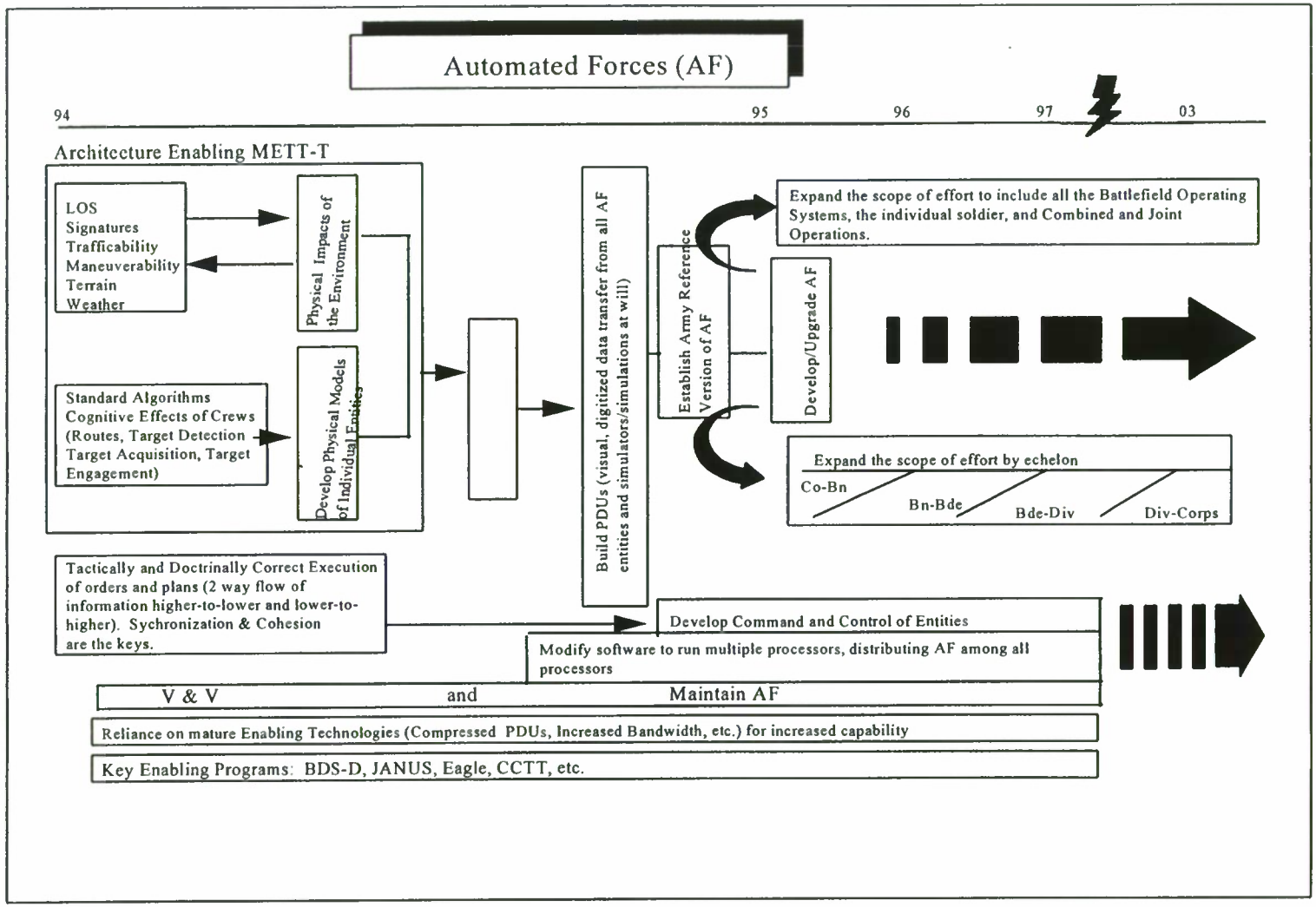


Figure 2 - Automated Forces

Finite state machines (FSMs) are often used to generate and control entity level behavior in CGF systems; ModSAF, the CCTT SAF, and the IST CGF Testbed all use variations of FSMs as a software control structure for their behavior generation algorithms (for descriptions of the FSM mechanisms, see (Calder,1993) for ModSAF, (Marshall,1994) for the CCTT SAF, and (Smith,1992) for the IST CGF Testbed).

Unfortunately, software FSMs do not lend themselves easily to the verification and validation process. Often, only those software engineers who have written the FSM for a particular system can validate the presence of a system characteristic. AMIP support should be given to a standardized FSM form for vehicular, personnel, and airframe representation. Dismounted infantry, being the most complex vehicular form to simulate, has had only first order looks at behavioral representations. Support should be given to projects improving infantry behavior and the physical environment (in particular micro terrain) supporting this behavior.

4.3 Command and Control of Virtual Entities

C2 of entities in most CGF systems are currently limited to Company and below. The behavior of entities (representing the ability of platform crews to locate battle positions, movement routes and engage targets) is driven by rule bases and the structure of the FSM controlling the entity. These behaviors perform reasonably well for simple tactical maneuvers. However, the control of units (squads, platoons and companies) performing combined maneuvers remains a difficult problem. Significant progress has been made in the development of the CCTT SAF in this area. Rule bases related to unit FMs and RTEPs are being developed and well documented. Through the use of these rule bases, CCTT SAF is targeted for representation up to a Brigade with SAF operators directing company units from their console. If CGF systems are expected to represent larger forces, as required by the DIS Master Plan, then an architecture must be established to represent the battle command process. At a minimum this architecture must include the following capability:

- A representation of the Battle Plan (in the form of an Operations Order) for distribution to all nits at each command echelon. CGF command units must have the ability to interpret both the battle plan and the control measures (objectives, phaselines, etc.) affecting their operation.

- A structure (possibly a limited battle language), for CGF command units to report to higher units the tactical state of lower echelon units executing the battle plan.

- The ability of higher echelon CGF command units to make decisions and manage the battle through the communication of commands and new battle plans to lower echelon units.

Work is being conducted by the DIS community in this area. Under ARPA sponsorship, MITRE is developing CCSIL (Command and Control Simulation Interface Language), to be used to communicate C2 commands and information between DIS nodes (Salisbury,1995).

Also under ARPA sponsorship, the CFOR project (Command Forces) is addressing the problem of high-level command for CGF entities. Proposals are still being accepted for CFOR. The goal of CFOR is to develop automated C2 and planning entities that fit into the command hierarchy. These CFOR entities would communicate with each other up and down the hierarchy, using CCSIL (MITRE has used the Eagle Management Language for the basis of this CCSIL effort.) The Corps Level Computer Generated Forces project (CLCGF) sponsored the Joint Precision Strike Demonstration (JPSD) will send Eagle-based orders to ModSAF companies where they will be acted on by ModSAF simulated company commanders. However there is no standard in this area. Further the two efforts mentioned above are focused on purely maneuver and indirect fire units. Little thought has been given to C2 requiring sensor tasking, on call support of fixed and rotary wing units, or support from engineer and logistic units. AMIP/DMSO support in this area is key if computer generated forces are ever to grow beyond the Battalion firefight.

4.4 Architecture for Distributed Computer Generated Forces

I have saved the most pressing problem for last. In 1992 when STRICOM published the original paper on the DIS architecture, they envisioned a system where remote sites could join a battle through nodes on the network. These sites represented users with different interests. Some were simulators of new weapon systems, others were CGF sites providing

forces for the network, still others were field equipment with live crews training in the virtual networked environment. The only requirement "to play" was conformance to the DIS Protocol Data Units. In short, if your site was sending and receiving the right message formats, interaction was possible.

Pursuing this vision, the DIS community has moved the DIS PDUs to an IEEE standard. DIS nodes have been established throughout the Army and Advanced Technical Demonstrations have been conducted placing more and more (up to 1000) virtual vehicles on the network. In the beginning, it was believed that the network bandwidth (the ability of the long haul and local networks to transmit the massive number of PDUs) would be the limiting factor on this system. However, as a result of problems encountered at recent ATDs and I/ITSEC demonstrations many believe that the limiting problem will stem from an inability of remote DIS sites to filter PDUs not affecting their simulator, simulation or live equipment.

To appreciate this problem, one must have an understanding of the tenets under which the DIS PDUs were developed. DIS has its ancestry in SIMNET. As such, three fundamental architectural features have governed development. When simulating any DIS vehicle you must:

- Broadcast ground truth from your entity/CGF simulation.
- Operate in an environment with distributed processors.
- Expect to broadcast and receive updates for dead reckoning algorithms to maintain locations of other vehicles being simulated on other processors.

Consider now the implication in terms of network packets from 1, 2 and 3. Every virtual platform (ground, air and sea) puts packets on the net describing its position, updates as it moves or performs other battle activities i.e. when a vehicle stops, turns its turret to fire, a packet indicating the stop goes onto the net and a packet indicating the turning of the turret (followed by many more as the turret turns and the gun lays) goes onto the net. Further, consider the implications of this structure in the environment of many participants in a large DIS exercise. Suppose we are at a node processing a CGF which is simulating the northern most Battalion taskforce in a Corps. As virtual tanks simulated by another CGF representing the southern most Battalion move and fire our node is required to examine each PDU from the south Battalion, determine that it is outside the virtual geographical area simulated by our node and discard it. In short, each participant under the current DIS architecture is globally broadcasting local information. The impact on each DIS processing node of filtering the global information from that information affecting the battle locally being represented at that node has become the major problem in DIS. This problem of "servicing the node", as it is called by those in the ATDs and L/ITEC experiencing it, will ultimately limit the number of players and the size of the battle in any exercise.

As this problem effects all DIS simulators, not just CGF systems, it has been taken up by the communications architecture group within the DIS community. Currently, the solution receiving the most attention is replacing broadcast with multicast. The idea is to use a large number (hundreds) of multicast groups created within a DIS exercise so that only those nodes that need to receive another node's PDUs are on its multicast group. This would allow a given simulator to only handle those packets it needs to, with the communications system filtering PDUs at the multicast level. The multicast group structure would be created and maintained automatically, possibly by a network services controller node. It is not yet clear what would be the best basis for creating the multicast groups; geographical proximity, sensor categories, and a data subscription request mechanism have all been suggested. Several papers discussing multicast in DIS were presented at the 12th DIS Workshop (e.g. (Calvin,1995) and (Pullen,1995)); an experimental test of applying multicast grouping based on geographical proximity to a CGF system is described in (Smith,1995).

A second problem is that the message structures do not address the integration of simulations and CGF systems representing varying levels of resolution. For example, suppose that a Corps level simulation was "on line" in a DIS exercise broadcasting ground truth locations of vehicles in Company units. Further, suppose a J-STARS simulator was displaying these vehicle traces. As the vehicles approach, a bridge or terrain constriction represented only as a "movement delay" in the Corps model, how will the J-STARS simulator effectively display the change in vehicular formation as the constriction is

negotiated? There must be an architecture structure developed for hand-off of control of these units if a consistent battle is to be represented to all participants. We realize that this is an interface problem but it clearly affects any standard developed for a CGF. Work addressing these issues has almost always depended heavily on CGF systems (Kraus, 1995).

The bottom line is that the overarching architecture for DIS, as reflected by the current DIS PDUs, is limited to the Brigade and lower battles. This structure will not satisfy the Army DIS Master Plan in either functionality or projected timelines. AMIP and DMSO must support research expanding/defining an architecture for a larger (Corps to Theater) DIS battle.

5. Summary of Recommendations/Priorities

The previous sections have described the requirements for developing a CGF system, related particular parts of the CGF system to the Army's DIS Master Plan, and summarized some problems with the DIS architecture relative to CGF systems. This section prioritizes the actions that should be taken to move toward a standard CGF system that will support all three application areas: training, advanced technology demonstrations, and analysis.

1. Adopt ModSAF as the standard CGF system for the short term.

Considerable developmental effort has been invested in ModSAF, it is widely distributed, and it has been subjected (at least in part) to VV&A procedures. Hence, it should be the standard CGF system, at least for now. In this light, two efforts become critical:

- VV&A of MODSAF as both a Training and Analytic CGF must be completed and given the necessary resourcing priorities.

- Improvements in environmental, terrain, C2 and DIS Architecture representation must be compatible and integrated into the ModSAF architecture.

2. Focus on the CCTT SAFOR for acceptance as the Standard CGF for the long term.

Again, this recommendation is based on the practicalities of resource consideration. The CCTT project is expensive ($10 million for the CGF alone) and the Army cannot afford to develop another standard. Hence as it comes on line, it must sustain a VV&A overhead that will make it acceptable for both training and combat developments uses.

3. Solve the DIS Architectural Problem.

The problems described under the section on the DIS architecture will ultimately affect all modeling efforts. In particular, CGF systems will be limited to Brigade and below. The WARSIM Project will not have a virtual capability and the DIS Master Plan will not extend beyond Brigade. While the DIS Master Plan lists "DIS Architecture/Networks" as condition green, the first author does not believe this to be the case. At best it is amber with the current structure not allowing robust expansion of the virtual battle field.

4. Solve the Environmental Representation Problem in CGF.

Advanced weapons technology is firmly based on sensor technology operating in spectral regions outside the visual. If we are unable to represent the simple effects of environmental attenuation for these sensors, how can we expect to train and test these systems? Any practical use of CGF demands a realistic representation of the battlefield environment and platform signatures.

6. Final Note

While this report has pointed out many of the problems currently being experienced by developers and users of CGF systems, we would be remiss if I did not close on a positive note. The Army software community (both the Training, Testing and Combat Developers) stands in the enviable position of leading DoD in CGF development. WE have no doubt that the technical problems listed above will be solved. It is just a matter of priority and focus. Further, we believe that the DIS and CGF structures will come into common usage in Training, Testing and Combat Developments Community within the next two years. This opinion comes from the experience and conservatism of one who has viewed the "blank page" in the course of simulation model development.

7. References

U. S. Army (1994). *Distributed Interactive Simulation Master Plan*, Training and Doctrine Command, Ft. Monroe, VA.

Booker, L., Brooks, P, Garrett, R., Giddings, V., Salisbury, M., and Worley, R. (1993). "1993 DMSO Survey of Semi-Automated Forces", *Defense Modeling and Simulation Office Report*, July 30 1993, 86 pages.

Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., and Ceranowicz, A. Z. (1993). "ModSAF Behavior Simulation and Control", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, March 17-19 1993, pp. 347-356.

Calvin, J. O., Seger, J., Troxel, G. D., and Van Hook, D. J. (1995). "STOW Realtime Information Transfer and Networking System Architecture", *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, IST-CF-95-01.1, Orlando FL, March 13-17 1995, pp. 343-353.

Ceranowicz, A. (1994). "ModSAF Capabilities", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 4-6 1994, pp. 3-8.

Franceschini, R. W. and Petty, M. D. (1995). "Linking constructive and virtual simulation in DIS", *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing & Control and Dual-Use Photonics*, Orlando FL, April 19-20 1995.

Kilby, M., Lisle, C., Altman, M., and Sartor, M. (1994). "Dynamic Environment Simulation with DIS Technology", *Proceedings of the 16th Interservice/Industry Training Systems and Education Conference*, Orlando FL, November 28-December 1 1994, pp. 4-18.

Kraus, M. K., Stober, D. R., Foss, W. F., Franceschini, R. W., and Petty, M. D. (1995). "Survey of Constructive+Virtual Models", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11 1995.

Marshall, H., Anderson, C., Baran, T., Berggren, P., Bimson, K., Blanchard, D., Braudaway, W., Burch, B., Colon, J., Cosby, J., Glover, G., King, J., Ourston, D., and Watson, J. (1994). "Close Combat Tactical Trainer Semi-Automated Forces (SAF) Design Overview", Presentation Notes, *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 4-6 1994.

Moshell, J. M., Blau, B. Li, X., and Lisle, C. (1994). "Dynamic Terrain", *Simulation*, Vol. 62, No. 1, January 1994, pp. 29-42.

Petty, M. D. (1992). "Computer Generated Forces in Battlefield Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, Pensacola FL, October 22-23 1992, pp. 56-71.

Petty, M. D. (1995). "Computer generated forces in DIS", *Proceedings of the SPIE International Symposium on Aerospace/Defense Sensing & Control and Dual-Use Photonics*, Orlando FL, April 19-20 1995.

Pullen, J. M. and White, E. L. (1995). "Dual-Mode Multicast for DIS", *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, IST-CF-95-01.1, Orlando FL, March 13-17 1995, pp. 505-510.

Salisbury, M. R. (1995). "Command and Control Simulation Interface Language (CCSIL): Status Update", *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, IST-CF-95-01.1, Orlando FL, March 13-17 1995, pp. 639-649.

Smith, J. E., Russo, K. L.,and Schuette, L. C. (1995). "Prototype Multicast IP Implementation in ModSAF", *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, IST-CF-95-01.1, Orlando FL, March 13-17 1995, pp. 175-178.

Smith, S. H., and Petty, M. D. (1992). "Controlling Autonomous Behavior in Real-Time Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola FL, October 22-23 1992, pp. 27-40.

Thomas, J. G. (1995). "Verification and Validation of Modular Semi-Automated Forces (ModSAF) in Support of A2ATD Experiment 1", *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, IST-CF-95-01.1, Orlando FL, March 13-17 1995, pp. 359-367.

8. Acknowledgements

The second author's contribution to this paper was supported by the U. S. Army Simulation, Training, and Instrumentation Command under the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged. The opinions stated herein, however, are those of the authors and do not necessarily reflect official STRICOM positions.

The authors thank Cindi Slepow for technical writing assistance with this paper

9. Authors' Biographies

H. Kent Pickett has over 20 years experience as an Operations Research Analyst. He has served as modeler and analyst at the Concepts Analysis Agency, and has held progressively higher analysis positions with the Department of Army at Fort Leavenworth, to include: Chief of the Special Studies Branch, TRADOC Analysis Operations Research Activity (TORA); Acting Director, Methodology and Quality Assurance Directorate, TRADOC Analysis Command (TRAC), and currently serves as Director, Modeling and Research Directorate, TRAC. Additionally, Mr. Pickett has 20 years of experience teaching computer science at Missouri Western State College, where he is an Assistant Professor of Computer Science. He received a B.S. degree in Mathematics and an M.S. degree in Applied Mathematics from the Universitry of Missouri-Rolla.

Mikel D. Petty is a Program Manager at the Institute for Simulation and Training. He is currently managing Plowshares, an emergency management simulation project. Previously he led IST's Computer Generated Forces research projects. Mr. Petty received a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from California State University, Sacramento. He is a Ph.D. student in Computer Science at UCF. His research interests are in simulation and artificial intelligence.

Asynchronous Rule-Based Systems in CGF

Paul F. Reynolds, Jr. and Craig Williams
reynolds@virginia.edu and ccw4s@virginia.edu
Dept. of Computer Science,
University of Virginia
Charlottesville, VA 22903

1. Abstract

CGF systems face two important challenges: first, behavioral and cognitive modeling must become more viable and, second, a means for efficiently implementing behavioral and cognitive models must be found. The first issue is difficult, and must be resolved by domain experts and modeling theoreticians. The second is difficult as well: the common implementation of behavioral and cognitive models is production (rule-based) systems, which are notoriously slow. We describe an alternative approach to parallel implementation of rule-based systems, employing *isotach networks*. Performance studies of isotach networks suggest they hold significant potential for order-of-magnitude speed-up for rule-based systems and, therefore, for behavioral and cognitive models for CGF.

2. Introduction

Currently, CGF systems (e.g. ModSAF, CCTT-SAF) employ finite state machines (FSM's) for modeling synthetic forces. As CGF requirements expand to include behavioral and cognitive models, the limitations of FSM's will force the consideration of more powerful modeling techniques. The most likely alternative is rule-based systems. Unfortunately, traditional approaches to the execution of rule-based systems have exhibited disappointing performance. Attempts to achieve significant speed-ups through parallel execution have been frustrated by synchronization overhead, in particular, by the match-recognize-act (MRA) cycle. We describe a new approach that eliminates the MRA cycle.

The first and most obvious improvement to a slow rule-based system is to optimize the code that implements the Rete network, the principal data structure of the rule-based system. However, optimization is only a start. As the numbers of rules and entities modeled increase, a point is quickly reached at which even an optimized system is inadequate. At this point, the hope for improving the performance of the rule-based system lies with parallel execution.

A rule-based system is composed of a set of if-then rules and a database of assertions called working memory elements (WME). A rule is eligible to fire (i.e., the actions in the "then" part of the rule can be executed) if the WME's match, i.e., satisfy, the conjunction of conditions that make up the "if" part of the rule. The most straightforward way to execute a rule-based system in parallel is to give each rule (or set of rules) its own processor. Each rule can then be evaluated in parallel. Since rule evaluation (i.e. trying to match rules against the WME's) accounts for about 90% of the execution time of a rule-based system, executing rules in parallel should be a winning strategy. However, parallel rule-based systems to date remain disappointingly slow. The problem is that rule firings must be coordinated in some way. Suppose two rules, R1 and R2, are both eligible to fire. Since firing a rule can change the WME's, firing R1 may make R2 ineligible and firing R2 may make R1 ineligible. Even though the "if" part of both rules is satisfied by the WME's, firing both R1 and R2 would be incorrect since the result would not be equivalent to any serial execution.

The conventional way to coordinate rule firings in parallel rule-based systems is the match-recognize-act cycle: each processor evaluates its rule(s) in parallel; the processors synchronize; a single rule is selected; the selected rule fires; and the cycle repeats. If each processor had roughly the same amount of work to do in each cycle, the MRA cycle would be a good way to coordinate rule firings. Unfortunately, firing a rule in a rule-based system tends to affect only a small number of other rules (Gupta et al. 1989). This phenomenon, known as the small cycle problem, means that only a small number of processors have any useful work to do in any given cycle. Thus the MRA cycle severely limits the number of processors that can be employed usefully in executing a rule-based system.

We eliminate the MRA cycle. We have found an alternative that provides the coordination needed to execute parallel rule-based systems correctly. Our

approach is based on a logical time system called *isotach networks* that can be implemented on arbitrary topologies and in both tightly-coupled multiprocessors and clusters of workstations. Isotach networks are characterized by the guarantee they provide about the relative order in which operations within the system appear to be delivered. The guarantees can be enforced cheaply, using purely local knowledge, and yet provide a sufficient basis for enforcing several important properties of parallel and distributed computations: causal message ordering, atomicity, and sequential consistency. The last two properties, especially atomicity, are important in rule-based systems. An atomic action is a group of operations that must appear to be executed as an indivisible step, i.e. without interleaving with operations by other processes. Atomicity is important in a rule-based system because correct execution requires that the "if" part of the rule be satisfied at the time the "then" side fires. In other words, each rule in a rule-based system is an atomic action. In an isotach rule-based system, rules fire asynchronously, i.e., a rule can fire whenever it is eligible. Processors with eligible rules do not need to synchronize before firing and yet, because each rule is executed atomically, the computation is correct: for each execution, there is an equivalent execution in which rules fire one at a time and no rule fires unless eligible.

In the following sections we discuss related approaches to logical time systems and parallel execution of rule-based systems. We describe isotach networks in more detail and results of performance studies on isotach networks. Finally, we describe our planned approach for applying isotach networks to rule-based systems and we discuss the utility of isotach technology to CGF.

3. Background

Two approaches to reducing the impact of the MRA cycle are being explored (Kuo, et al. 1992). One approach is to reduce the granularity of the computation by partitioning the Rete network (Forgy, 1982) among the processors. Reducing the granularity of the computation makes it possible, with efficient scheduling, to reduce the variance among processor workloads in each cycle, but it also increases synchronization overhead. The other approach is to select multiple rules to fire per cycle: each processor evaluates its rule; the processors synchronize; and a set of non-conflicting rules is selected for firing. This approach allows more processors to do useful work during the parallel match phase of the cycle, but the need to identify a set of non-conflicting rules increases the length of the sequential phase of the cycle.

We are not the first to propose eliminating the MRA cycle. Schmolze proposed an asynchronous system (Schmolze et al. 1990) in which the coordination among rules is handled by the same techniques used to enforce atomicity in distributed databases: locking, using a linear ordering among rules to prevent deadlock. Our approach differs from this proposal in that it eliminates the MRA cycle without introducing locking. We expect the isotach system to perform significantly better than this earlier asynchronous system because isotach systems are more efficient than conventional systems in providing synchronization.

By abandoning the MRA cycle, our approach employing isotach systems removes the imposition of sequential control on an inherently parallel process. Isotach systems are based on logical time. The seminal paper by Lamport (Lamport 1978) established a basis for isotach-like timing systems. Other systems similar to isotach systems have been proposed for other purposes by Awerbuch (Awerbuch 1985), Ranade (Ranade 1987), and Birk (Birk et al. 1989). Isotach systems are unique in providing guarantees of atomicity, sequential consistency *and* causal message ordering.

The performance of isotach systems has been studied through extensive simulation (Reynolds, 1992). Order of magnitude speed-ups have been observed for a variety of workloads in which atomicity and the opportunity to exploit pipelining (simultaneous issuing of multiple memory references by the same process) are present.

4. Isotach Systems

4.1 Details of Isotach Technology

Isotach networks are characterized not by their topology —they can be implemented on arbitrary topologies and on clusters of workstations as well as on tightly-coupled multiprocessors— but by the guarantee they provide about the relative order in which operations appear to be delivered. The guarantee is expressed in logical time (an extension of Lamport's logical time), not physical time, i.e., the guarantee concerns the relative order in which messages *appear* to be delivered. Except in the case of messages delivered to the same place, the order in which messages appear to be delivered is not necessarily the order in

which they are actually delivered. Physical time guarantees would be prohibitively expensive, whereas logical time guarantees can be enforced cheaply, using purely local knowledge, and yet provide a sufficient basis for enforcing atomicity without locks and sequential consistency without restricting pipelining of operations.

In isotach networks, each message progresses towards its destination at the same rate: one unit of logical distance (for our purposes here, one hop within the network) per logical time unit. This property of isotach networks, called the *velocity invariant*, implies that the logical time at which an operation is emitted into the network completely determines the logical time at which it is received and executed. Thus a process can control the logical times at which its operations are executed by controlling the logical times at which they are emitted. This control over logical execution time allows each process to ensure that its accesses are executed in an order consistent with the program's atomicity and sequencing constraints without synchronizing with other processes or waiting for acknowledgments from memory.

In the remainder of this section we discuss details of isotach networks. The reader may wish to proceed to the section on applications and then return to this point when a deeper understanding of the technology underlying isotach networks is desired.

Assigning logical times to events is a way of ordering events. A logical time system is a set of constraints governing the way in which events of interest are ordered, i.e. assigned logical times. Isotach logical time is an extension of the logical time system defined by Lamport in his classic paper on ordering events in a distributed system (Lamport 1978). Lamport's time system assigns times consistent with the *happened before* relation, a relation over the events of sending and receiving messages that captures the notion of potential causality: event *a happened before* event *b*, denoted $a \rightarrow b$, if 1) *a* and *b* occur at the same process and *a* occurs before *b*; 2) *a* is the event of sending message *m* and *b* is the event of receiving the same message *m*; or 3) there exists some event *c* such that $a \rightarrow c$ and $c \rightarrow b$. In Lamport's system $a \rightarrow b \Rightarrow t(a) < t(b)$, where for any event *x*, $t(x)$ denotes the logical time assigned to *x*.

Lamport gives a simple distributed algorithm that maintains this time system. Each process has its own logical clock, a variable that records the time assigned to the last local event. When it sends a message, a process increments its clock and timestamps the message with the new time. When it receives a message, a process sets its clock to one more than the maximum of its current time and the timestamp of the incoming message. This algorithm ensures $a \rightarrow b \Rightarrow t(a) < t(b)$. Several researchers (Mattern 1988, Schmuck 1988, Fidge 1988) have independently described a way to maintain the more stringent constraint $a \rightarrow b \Leftrightarrow t(a) < t(b)$ by using vectors of logical times. Each element in a vector represents a logical time at one processor. This stricter form of Lamport's logical time system has been used to implement communication primitives for distributed computation (Birman et al. 1991).

In an isotach logical time system, logical times are lexicographically ordered n-tuples of integers of which the first and most significant component is called the *pulse* component. Unless otherwise stated, assume logical times are 3-tuples in the form $(pulse,x,y)$. Isotach logical time extends Lamport's logical time by imposing the additional constraint that logical times be consistent with the *velocity invariant:* each message *m* is received exactly $d(m)$ pulses after it is sent, where $d(m)$ denotes the logical distance *m* travels. For any message *m*, let $ts(m)$ denote the logical time assigned to the send event for *m* and $tr(m)$ the logical time assigned to the receive event for *m*. Thus for any message *m*, $ts(m) = (i,j,k) \Rightarrow tr(m) = (i+d(m),j,k)$. Isotach networks are so named because all messages travel at the same velocity in logical time —one unit of logical distance per pulse. Unless otherwise stated, logical distance, $d(m)$, is simply the number of (possibly virtual) switches through which *m* is routed.

The velocity invariant requires that the logical time assigned to the receive event for a message that travels zero distance be the same as the logical time assigned to its send event. To accommodate this case in which the source and destination of a message are collocated, we have relaxed Lamport's requirement that the time assigned to event *a* that potentially causes event *b* be strictly less than the time assigned to event *b*. In isotach logical time, $a \rightarrow b \Rightarrow t(a) \leq t(b)$.

The number and interpretation of the components that follow the pulse component can vary. In this paper, each logical time is of the form $(pulse, pid(m), rank(m))$, where for any message *m*, $pid(m)$ is the identifier of the process that issued *m* and $rank(m)$ is the issue rank of the message, i.e., $rank(m) = i$ if *m* is the i^{th} message issued by $pid(m)$. In a database system the *pid* may be replaced by a transaction identifier. In some isotach systems, for example in a system that

combines operations on the same shared variable within the network, a 4-tuple isotach logical time system is most appropriate. In a 4-tuple system each logical time is in the form (pulse, destination, source, rank), where the destination component names the variable accessed (shared memory model) or the destination process or object (message-based model).

4.2 APPLICATIONS

In a system that maintains isotach logical time, a processor (PE) can control the logical time at which the messages it sends are received and executed. (Note: "sending messages" may be simple memory references, for example, a single shared memory reference.) This ability to control the logical time of remote events is the basis for new techniques for enforcing atomicity and sequential consistency. This section describes these techniques.

An *atomic action* is a group of one or more instructions issued by the same process that appears to be executed indivisibly, i.e., without interleaving with other instructions (Owicki et al. 1976, Lomet 1977). In some contexts, in particular in databases, the atomic action is also a unit of recovery from hardware failure. We assume the system is fault-free or that faults are handled below the application level. Conventional systems enforce atomicity with some form of locking. Drawbacks of locking include overhead for lock maintenance in space, time, and communication bandwidth, unnecessarily restricted access to shared variables, and the care that must be taken when using locks to avoid deadlock and livelock. In an isotach system, a process can execute *flat* atomic actions, (atomic actions containing no internal data dependences among shared variables) without synchronizing with other processes and can execute *structured* atomic actions (atomic actions with such dependences) without acquiring locks or otherwise obtaining exclusive access rights to the variables accessed.

An execution is *sequentially consistent* if the order in which operations are executed is consistent with the order specified by each individual process's sequential program (Lamport 1979). This ordering guarantee is so basic it is easily taken for granted, yet it is expensive to enforce. The problem is that the order in which operations are received and executed may differ from the order in which the operations were sent into the network due to stochastic delays within the network. The conventional solution is to prohibit pipelining, meaning that each process must delay issuing an operation until it receives notice that its preceding operation has been executed. Since pipelining is an important way to decrease effective memory latency, this solution is expensive. The high cost of enforcing sequential consistency has led to extensive exploration of weaker memory consistency models, e.g., (Scheurich et al. 1987, Gharachorloo et al. 1991). These weaker models are harder to reason about and still impose significant restrictions on pipelining, but they make sense given the cost of maintaining sequential consistency in a conventional system. In an isotach system, processes can pipeline memory operations without violating sequential consistency.

The velocity invariant is the key to enforcing atomicity and sequential consistency in an isotach system. Given the velocity invariant, a PE that knows $d(m)$ for each operation m it sends can control the logical time at which its operations are received and executed by its choice of the logical time at which it sends the operations. A PE can enforce atomicity and sequential consistency by following these rules:

ATOMICITY. Send operations from the same flat atomic action so they are received in the same pulse.

SEQUENTIAL CONSISTENCY. Send each operation so it is received in a pulse no earlier than that of the operation issued before it.

The rules are applicable to any topology, but are especially easy to apply in equidistant networks: a PE sends all operations in the order in which they are issued and all operations from the same flat atomic action in the same pulse.

Example. Consider the non-equidistant network in Fig. 1, in which each oval represents a switch and each rectangle a memory module (MM) or PE:

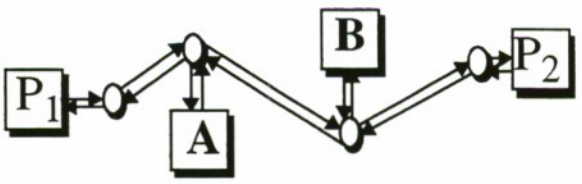


Fig. 1. A Non-Equidistant Network.

Suppose *p1* is required to read shared variables *A* and *B* atomically and *p2* to write *A* and *B* atomically. In conventional systems, *p1* and *p2* need to obtain locks, either on the individual variables or on a section of code (as in a critical section), to ensure its accesses are executed atomically. In isotach systems, *p1* and *p2* can execute their accesses without synchronizing, as follows: *p1* sends the operation on *A* one pulse after it sends the operation on *B* and *p2* sends the operation on *B* one pulse after it sends the operation on *A*. By the velocity invariant, both operations in each atomic action are received in the same pulse. If all four operations happen to be received in the same pulse, operations on each shared variable will be received and executed in order by *pid*. Thus each atomic action will appear to be executed without interleaving with other operations. It is possible that operations may be executed in an interleaved order in physical time. For example, the actual execution order may be *p1*'s read to *A*, *p2*'s write to *A*, *p1*'s read to *B*, and *p2*'s write to *B*. This execution is correct nonetheless because it is equivalent to a serial execution in which operations in each atomic action are executed without interleaving: *p1*'s read to *A*; *p1*'s read to *B*; *p2*'s write to *A*; and *p2*'s write to *B*.

We have also described isotach-based techniques for executing structured atomic actions (Williams 1993). Structured atomic actions cannot be executed in the same way as flat atomic actions because data dependences among operations make it impossible to issue all the operations in a batch, but the techniques for executing flat atomic actions together with a class of operations called *split operations* support execution of structured atomic actions.

We have proposed several techniques for executing structured atomic actions using the isotach network in combination with split operations. Split operations are used to execute read and write accesses in two steps: the first schedules the access and the second transfers a value. The advantage of dividing accesses into two steps is it allows a process that has incomplete knowledge about an access due to an unsatisfied data dependency to reserve a slot in the variable's history that ensures it will appear to be executed at the same time as the other operations in the atomic action. (Another advantage is that it provides a mechanism for enforcing inter-process sequencing constraints.) An unsubstantiated write (a write for which the first step, but not the second has been executed) delays completion of reads scheduled up to the next write until the write is substantiated, but does not delay writes or other reads to the same variable.

A process executes a structured atomic action by issuing a batch of split operations scheduling all the accesses required for the atomic action, executing the assignment steps for these accesses as it determines the values to be assigned. Execution is atomic because the set of operations used to schedule the accesses reserves a consistent *time slice* across the histories of the accessed variables. This technique works for atomic actions with access sets that can be determined at the beginning of execution of the atomic action. We have proposed variations on the technique for atomic actions with data dependent access sets (Williams et al. 1989). Isotach based techniques for executing flat and structured atomic actions extend to systems with caches. In fact, extending the techniques to systems with caches eliminates the need for a separate type of memory otherwise required to support structured atomic actions.

4.3 Performance

We have completed both analytical (Wagner 1993) and empirical (Reynolds 1992) studies of isotach systems. We simulated conventional and isotach systems under a variety of workloads. Our results show that conventional systems have higher raw power, i.e., more throughput with less delay, than isotach networks, but that under workloads that include atomicity and sequencing constraints, isotach networks outperform conventional networks. The studies show order of magnitude performance improvement under realistic workloads. In extreme cases of high contention for shared objects, conventional systems cease to perform acceptably, but isotach systems continue to perform well.

5. Isotach, Rules, and CGF

5.1 Isotach and Rule-Based Systems

Atomicity and sequential consistency, especially atomicity, are important in rule-based systems. An atomic action is a group of operations that must appear to be executed as an indivisible step, i.e. without interleaving with operations by other processes. Atomicity is important in a rule-based system because correct execution requires that the "if" part of the rule be satisfied at the time the "then" side fires. In other words, each rule in a rule-based system is an atomic action. In an isotach rule-based system, rules fire asynchronously, i.e., whenever eligible. Processors with eligible rules do not need to synchronize before firing and yet, because each rule is executed atomically, the computation is correct: for each execution,

there is an equivalent execution in which rules fire one at a time and no rule fires unless eligible. Eliminating the MRA cycle should make it possible to exploit all or most of the rule-level parallelism available in the application.

Schmolze and Goel (Schmolze et al. 1990) proposed an asynchronous system in which coordination among rules is handled by the same techniques used to enforce atomicity in distributed databases: locking, using a linear ordering among rules to prevent deadlock. Our method differs in two respects:

- *The method of coordinating rule firings.* An isotach rule-based system eliminates the MRA cycle without introducing locking. We expect the isotach system to perform significantly better than asynchronous systems that achieve atomic execution of rules through locking because isotach systems are many times (order of magnitude) more efficient than conventional systems in enforcing atomicity (Reynolds et al. 1992).
- *The granularity of the computation.* We intend to exploit fine-grained parallelism as well as rule-based parallelism. Other researchers have investigated distributing the Rete network, the principal data structure used in a rule-based system, over multiple processors. The previous work on using fine-grained parallelism has all taken place within the context of the MRA cycle, so this is a new challenge. It is important to consider exploiting fine-grained parallelism in rule-based systems because it increases available parallelism and because relaxing the MRA cycle alone is not of help with rule-based systems in which the conditions for rules implicitly require that they be fired serially.

We discussed split operations in section 4.2. A system of rules could be executed concurrently, employing split operations, in the manner described below. Each process representing one or more rules would perform the following steps asynchronously:

1. Read the WME's that potentially match the rule's left-hand-side and schedule writes called for by the rule's right-hand-side operations. All of these reads and writes would be scheduled to occur at the same logical time. Note, there is no requirement that components of the WME be resident on a single processor. Using isotach-based timing, accesses will occur at the same logical time even on widely distributed networks.

2. Perform the match step for the rules. This step would be highly parallel, with each process doing its match independent of the activities of other processes.

3. For each rule that can fire, the process can write any WME's required by the right-hand-side of the rule.

4. For each rule that can't fire, the process can cancel the writes it just scheduled for the right-hand-side of the rule.

These four steps would be executed repeatedly by a process as long as it had rules to evaluate. Note, there is no direct, explicit synchronization among processes. There are no locks. Rules fire as quickly as they can be determined eligible to fire. The only form of synchronization that occurs between processes is when one rule causes writes to be scheduled for WME's that need to be read by a second rule. Analysis of the second rule would be delayed only until the process for the first rule either substantiated the writes or cancelled them. *This is the only synchronization that takes place in the analysis and firing of rules.*

A pessimistic variation of steps 1 and 2 given above would be:

1. Read the WME's for the match step, and perform matching.

2. If a match occurs, re-read the left-hand-side data and schedule writes called for by the rule's right-hand-side operations (all at the same logical time).

This pessimistic variation has the advantage of reducing contention by performing the match phases without scheduled writes pending. Its primary disadvantage is that it could force rule evaluation to take longer. Choice of the two alternatives could be adaptive depending on contention among rules for reading and writing WME's.

Some production systems also use meta-rules (e.g. SOAR (Laird, 1990)): rules to select among eligible rules. Meta-rules are anathema to the isotach approach because they assume an eligible set of rules is identified and then an MRA-like process is applied where one rule to fire is selected (using the meta rules for guidance). We are exploring ideas for incorporat-

ing preferences and/or constraints expressed in meta-rules within an isotach system. The pessimistic variation of steps 1 and 2 above is one example of how this might be done: a low priority rule must use the pessimistic approach while higher priority rules can use the more aggressive approach.

5.2 Isotach and CGF

Eliminating the MRA cycle should make it possible to exploit all or most of the parallelism available in the application. In other words, we would expect to see the speed of the system continue to improve as more processors are added until a limit inherent in the application is reached. Conventional parallel rule-based systems, by contrast, are limited by synchronization overhead to a low level of parallelism. Isotach rule-based systems should be well suited to CGF because we expect such applications to be inherently highly parallel due to the large number of entities possessing some scope for independent action.

For example, in a tank platoon, the independent actions within the tank would include terrain reasoning, situational assessment and planning. Actions coordinated at the platoon level would include activities both within the platoon itself and with echelons above the platoon, including communication and receiving and processing orders.

Within a C^4I hierarchy we would expect significant independent processing at individual nodes within the hierarchy and occasional coordination among nodes to carry out a common mission.

Even if we are mistaken in our assessment of independence, isotach technology is significantly better able to manage contention than other existing concurrency control techniques. Thus whether entities exhibit significant independence or not, isotach technology should outperform other approaches.

A simulation study of isotach-based production systems is underway employing synthetic workloads. As good rule bases become available, we are considering switching to trace-driven simulation.

6. Conclusion

Isotach networks offer an efficient approach to ordering rule analysis and firings without explicit synchronization. Isotach networks are logical networks that can be embedded in any physical network, ranging from high performance parallel computer backplanes to distributed networks of workstations connected by ATM's.

The traditional MRA-based approach to rule analysis using parallel computation is severely flawed: there is a sequential bottleneck in the middle of the MRA cycle. Our approach using isotach networks abandons the MRA cycle and bases rule analysis, selection and firing on isotach-based logical timing. Performance analysis of isotach networks is very encouraging, displaying order of magnitude speed-up over networks lacking isotach timing in cases where reasonable amounts of atomicity and pipelining were present. Rule-based systems have significant potential to exploit isotach technology because they have an abundance of atomic actions (multiple, concurrent read-write sets from individual rules). CGF-oriented rule sets should be particularly amenable to isotach technology because we expect such applications to be inherently highly parallel due to the large number of entities possessing some scope for independent action.

A simulation study is currently underway to study the performance benefits isotach networks will bring to CGF-oriented rule-based systems. We will report on that study in a later paper.

7. Acknowledgments

We thank Janet Morrow of the National Ground Intelligence Center for her support in this research effort. This research has been supported in part by DUSA(OR) funds in simulation technology.

8. References

Awerbuch, B. (1985) Complexity of Network Synchronization, *J. ACM*, 32,4, 804-823.

Birk, Y., Gibbons, P.B., Sanz, J.L.C., and Soroker D. (1989) A Simple Mechanism for Efficient Barrier Synchronization in MIMD Machines, *Tech. Rep. RJ 7078* (67141), IBM.

Birman, K.P., Schiper, A., and Stephenson, P. (1991) Reliable Communication in the Presence of Failures, *ACM Transactions on Computer Systems*, 5,1, 47-76.

Fidge, C. (1988) Timestamps in Message-Passing Systems that Preserve the Partial Ordering, *Proc. 11th Australian Computer Science Conference*, 56-66.

Forgy, C.L. (1982) RETE: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem, *Artificial Intelligence*, 19, 17-37.

Gharachorloo, et al. (1991) Performance Analysis of

Memory Consistency Models for Shared-Memory Multiprocessors, *4th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 245-257.

Gupta, A., Forgy, C., and Newell (1989) High-Speed Implementations of Rule-Based Systems, *ACM Transactions on Computer Systems*, 7,2, 119-146.

Kuo, S. and Moldovan, D. (1992) The State of the Art in Parallel Production Systems, *Journal of Parallel and Distributed Computing*, 15, 1-26.

Laird, J., et al. (1990) Soar User's Manual Version 5.2, *Tech. Report CMU-CS-90-179*. Carnegie Mellon Univ., Pittsburgh, PA, 1990.

Lamport, L. (1978) Time, Clocks, and the Ordering of Events in a Distributed System, *Communications of the ACM*, 21, 7, 558-565.

Lamport, L. (1979) How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs, *IEEE Trans. on Computers*, 28, 690-691.

Lomet, D.B. (1977) Process Structuring, Synchronization, and Recovery Using Atomic Actions, *Proc. Conf. on Language Design for Reliable Software, SIGPLAN Notices* 12,3, 128-137.

Mattern, F. (1988) Virtual Time and Global States of Distributed Systems, *Parallel and Distributed Algorithms*, 215-226.

Owicki, S. and Gries, D. (1976) An Axiomatic Proof Technique for Multiprocessor Systems I, *Acta Informatica* 6, 319-340.

Ranade, A.G. (1987) How to Emulate Shared Memory, *IEEE Annual Symposium on Foundations of Computer Science*, Los Angeles, 185-194.

Reynolds, Jr., P.F., Williams, C. and Wagner, Jr., R.R., (1992) Empirical Analysis of Isotach Networks, *Tech. Rep. 92-19*, University of Virginia, Department of Computer Science.

Scheurich, C. and Dubois, M. (1987) Correct Memory Operation of Cache-Based Multiprocessors, *Proc 14th Int. Symp. Computer Architecture*, 234-243.

Schmolze, G.H. and Goel, S. (1990) A Parallel Asynchronous Distributed Production System, *Eighth National Conf. on Artificial Intelligence*, 65-71.

Schmuck, F. (1988) *The Use of Efficient Broadcast in Asynchronous Distributed Systems*. Ph.D. Thesis, Cornell University.

Wagner, Jr., R.R. (1993) *On the Implementation of Local Synchrony*. Ph.D. Thesis, University of Virginia.

Williams, C.C. (1993) *Concurrency Control in Asynchronous Computations*. Ph.D. Thesis, University of Virginia.

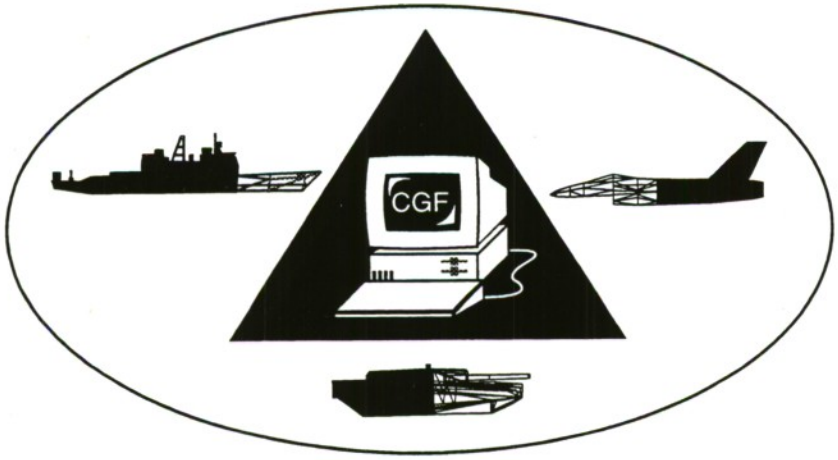
9. Authors' Biographies

Paul F. Reynolds, JR., Ph.D., University of Texas at Austin, '79, is an Associate Professor of Computer Science at the University of Virginia. He has been a member of the faculty at UVa since 1980. He has published widely in the area of parallel computation, specifically in parallel simulation, and parallel language and algorithm design. He has served on a number of national committees and advisory groups as an expert on parallel computation, and more specificality an expert on parallel simulation. He has been a consultant to numerous corporations and government agencies in the systems and simulation areas.

Craig Williams received her B.A. degree in Economics from the College of William and Mary in 1973, a J.D. from Columbia Law School in 1977, and her Master's degree in Computer Science in 1987. Her Ph.D. was awarded by the University of Virginia in 1993. She is now on the research staff in the Computer Science Department at the University of Virginia. Her research interests include parallel data structures and both the hardware and software aspects of parallel computation.

Author's List

Alhers, Robert - 53
Anderson, Chuck
Berggren, Peter
Bimson, Dr. Kent - 203
Blanchard, David - 275
Booker, Lashon B. - 423
Bowden, Fred D.J. - 513
Braudaway, Dr. Wes
Breneman, Larry J. - 443
Buettner, Cedric - 399
Burch, Bob
Buller, Mark J. - 487
Calder, Robert B. - 71, 83
Campbell, Chuck - 233
Ceranowicz, Andy - 3, 135
Chamberlain, Forrest - 83, 399
Chandler, Edward - 275
Christenson, W. M. - 501
Cisneros, Jaime E. - 159, 245
Courtemanche, Anthony J. - 3, 267
Craft, Micheal A. - 245, 433, 451
Crowe, Mike - 329
Dahmann, Judith S. - 423
Davies, Mike - 15, 513
D'Errico, John - 501
Dunn, John M. - 513
Evans, Dr. Alan - 399
Foss, William F. - 93
Franceschini, Robert W. - 21, 93, 103, 315, 465
Fuller, J. Mark
Gabrisch, Carsten - 15
Gagné, Denis - 521
Ge, Xiaolin - 45
Geib, Christopher - 345
Gonzalez, Avelino - 53
Hamilton, Scott E. - 267
Hieb, Michael R. - 135, 355
Hille, David - 135, 355
Hoff, Bruce - 255, 479
Howard, Micheal D. - 255, 479
Hoyt, Reed W. - 487
Hull, Richard - 233
Jackson, Lance - 233
James, John - 45
Johnson, W. Lewis - 27
Johnson, Thomas E. - 71
Jones, Randolph M. - 27
Karr, Clark R. - 159, 245, 295, 443, 451
Kelly, Joseph - 385
Kendall, Gary - 149
Koc, Nazim - 63
Kocabas, Sakir - 63
Koss, Frank - 27
Kraus, Matthew K. - 93
Kwak, Se-hung - 529
Laird, John E. - 27
Lankester, Helen - 169
Lehman, Jill Fain - 27, 115
Loh, Elsie - 275
Longtin, Michael - 305
Mall, Howard - 203
Marshall, Henry - 275
Mastroianni, George R. - 487
McAndrews, Gary - 219
McCormack, Jenifer - 203
McGhee, Robert - 219, 287
McIntyre, Robert T. - 391, 495
Megherbi, Dalila - 305
Meliza, Larry L. - 181
Mellies, Penny - 543
Metzler, Theodore - 385
Middleton, Victor E. - 495, 501
Mohn, Howard - 287
Monday, Paul - 193, 267
Moore, Michael B. - 345
Nee, Hai-Lin - 159
Nerode, Anil - 45
Nielsen, Paul E. - 27, 211
O'Keefe, IV, John A. - 391
Ourston, Dirk - 203, 275
Oztemel, Ercan - 63
Page, Ian - 149
Panagos, James - 71, 83
Pandari, Ashok - 367
Peacock, Jr., Jeffrey C. - 71, 83
Perneski, James - 193
Petty, Mikel D. - 93, 103, 315, 337, 433, 451, 549
Pickett, H. Kent - 549
Pratt, David R. - 219, 287
Pullen, J. Mark - 135, 355
Rajput, Sumeet - 159, 295, 443, 451
Reich, Barry D. - 345
Reynolds, Paul F. , Jr. - 559
Root, Eric - 233
Rosenbloom, Paul S. - 27, 39, 125
Rubinoff, Robert - 27, 115
Salisbury, Marnie R. - 423
Schaper, Gregory A. - 367
Schricker, Stephen A. - 315, 465
Schwamb, Karl - 27, 39
Seidel, David W. - 423
Shillcutt, Don - 329
Slepow, Mary P. - 337
Smith, Joshua E. - 375
Stanzione, Thomas - 83, 399
Stober, David R. - 93
Tambe, Milind - 27, 39, 125
Tecuci, Gheorghe - 135, 355
Thomas, John G. - 197
Tolley, Tracy R. - 315, 465
Tseng, David Y. - 255, 479
Uldudag, Mahmul - 63
Vaden, Eric A. - 181
Van Dyke, Julie - 27, 115
vanLent, Michael - 27
Warren, Richard - 329
Watkins, Jon - 411
West, Paul D. - 337
Williams, Craig - 559
Wise, Ben P. - 83
Wray III, Robert E. - 27


CGF



University of Central Florida · Division of Sponsored Research
Institute for Simulation & Training · 3280 Progress Drive · Orlando, Florida 32826