IDA DOCUMENT D-1484

CCTT/CATT SAFOR PANEL DISCUSSIONS 27-29 OCTOBER 1993

Philip Anton
Peter Brooks
Paul Gorman
John Laird
Duncan Miller
Robert Richbourg
J. D. Fletcher

February 1994

Prepared for
Advanced Research Projects Agency

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

Series B
IDA Log No. HQ 94-45111

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

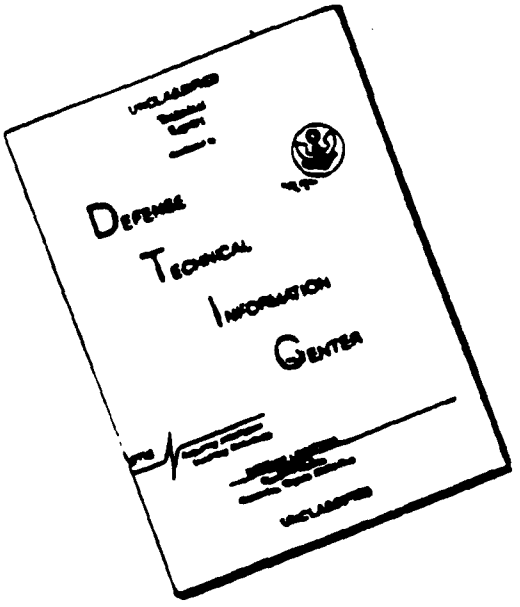
Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1994	Final--27–29 October 1993

4. TITLE AND SUBTITLE

CCTT/CATT SAFOR Panel Discussions 27–29 October 1993

5. FUNDING NUMBERS

C - MDA 903 89 C 0003

T- ARPA Assignment A-132

6. AUTHOR(S)

Philip Anton, Peter Brooks, Paul Gorman, John Laird,
Duncan Miller, Robert Richbourg, J.D. Fletcher

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

Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria. VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER

IDA Document D-1484

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 N. Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 180 words)*

Semi-Automated Forces (SAFOR) are a key component of the Distributed Interactive Simulation (DIS) environment. A SAFOR capability is being developed for the Close Combat Tactical Trainer (CCTT) production program, which is part of the larger Combined Arms Tactical Trainer (CATT) effort. Panel discussions were held 27–29 October 1993 on the development of CCTT/CATT SAFOR and its ability to exchange ideas and products with all DIS programs. The panel concluded that the widest possible community should develop and share ownership in a CCTT/CATT SAFOR product. More specifically: (1) The same SAFOR products can and should be used to support both the research and development and the user community; (2) Two computationally separate SAFOR lines of development, one based on an Ada environment and one based on a C environment, would inevitably develop discrepancies, become insufficiently coordinated, and should not be pursued; (3) The research and development community and other interested communities are unlikely to have either the resources or inclination to migrate to an Ada programming environment; (4) Products from CCTT/CATT SAFOR development should be made as accessible and adaptable as possible--higher priority should be given to accessibility and adaptability than to life-cycle maintainability; (5) CCTT/CATT SAFOR development should be pursued using a C and/or C++ programming environment.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Semi-Automated Forces (SAFOR); Close Combat Tactical Trainer (CCTT); Combined Arms Tactical Trainer (CATT); Distributed Interactive Simulation (DIS)	434
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

IDA DOCUMENT D-1484

CCTT/CATT SAFOR PANEL DISCUSSIONS
27-29 OCTOBER 1993

Philip Anton
The Mitre Corporation

Peter Brooks
Institute for Defense Analyses

Paul Gorman
Cardinal Point, Inc.

John Laird
University of Michigan

Duncan Miller
MIT Lincoln Laboratory

Robert Richbourg
U.S. Military Academy, West Point

J. D. Fletcher
Institute for Defense Analyses

February 1994

INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 89 C 0003
ARPA Assignment A-132

ABSTRACT

Semi-Automated Forces (SAFOR) are a key component of the Distributed Interactive Simulation (DIS) virtual environment. A SAFOR capability is being developed for the Close Combat Tactical Trainer (CCTT) production program, which is part of the larger Combined Arms Tactical Trainer (CATT) effort. Panel discussions were held 27-29 October 1993 on the development of CCTT/CATT SAFOR and its ability to exchange ideas and products with all DIS programs. The panel concluded that the widest possible community should develop and share ownership in a CCTT/CATT SAFOR product. More specifically: (1) The same SAFOR products can and should be used to support both the research and development and the user community; (2) Two computationally separate SAFOR lines of development, one based on an Ada environment and one based on a C environment, would inevitably develop discrepancies, become insufficiently coordinated, and should not be pursued; (3) The research and development community and other interested communities are unlikely to have either the resources or inclination to migrate to an Ada programming environment; (4) Products from CCTT/CATT SAFOR development should be made as accessible and adaptable as possible—higher priority should be given to accessibility and adaptability than to life-cycle maintainability; (5) CCTT/CATT SAFOR development should be pursued using a C and/or C++ programming environment.

CONTENTS

Abstract iii

Abbreviations and Acronyms vii

OVERVIEW 1

A. Background 1

B. Findings 4
1. Immaturity of SAFOR Technology 5
2. Coordination with Research and Development 5
3. Breadth of the SAFOR Requirement 6
4. Re-engineering of ModSAF 7
5. Quality of Software Engineering 7
6. Commercial Tools 7
7. National Guard Requirements 7
8. C and C++ Programming Environments 8

C. Recommendations 8

D. Conclusions 9

Appendix A-- Panel Members' Comments A-1

Appendix B-- Biographical Sketches of Panel Members B-1

Appendix C-- Survey of Semi-Automated Forces Briefing (MITRE Corporation) C-1

Appendix D-- ModSAF: An Overview Briefing (Loral Corporation) D-1

Appendix E-- Computer Generated Forces (Institute for Simulation and Training) E-1

Appendix F-- Automated Forces Technology Program Briefing (Advanced Research Projects Agency) F-1

Appendix G-- SAF Concurrent Engineering Team Briefing (IBM Corporation) G-1

ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
ARPA	Advance Research Projects Agency
ASRMO	Army Software Reuse Management Office
BDS-D	Battlefield Distributed Simulation-Developmental
21 CLW	21st Century Land Warrior
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CATT	Combined Arms Tactical Trainer
CCSIL	Command and Control Simulation Inter
CCTT	Close Combat Tactical Trainer
CGF	Computer Generated Forces
CIS	Combat Instruction Set
CLIPS	C Language Integrated Production System
COOL	CLIPS Object-Oriented Language
COTS	Commercial Off-The-Shelf
CSCI	Computer Software Configuration Items
CTC	Combat Training Center
DDR&E	Director of Defense Research and Engineering
DI	Dismounted Infantry
DIS	Distributed Interactive Simulation
DISC4	Director of Information Systems for Command, Control, Communications, and Computers
DMSO	Defense Modeling and Simulation Office

DoD	Department of Defense
FDDI	Fiber-Optic Data Distribution Interface
GOTS	Government Off-The-Shelf
IDA	Institute for Defense Analyses
IDD	Interface Definition Document
IDS	Interface Design Specification
IDT	Integrated Development Team
IFOR	Intelligent Forces
I-PORT	Individual Portal (into Distributed Interactive Simulation)
IR	Infrared
IST	Institute for Simulation and Training (University of Central Florida)
LADS	Loral Advanced Distributed Simulation
MCC	Microelectronics and Computer Technology Corporation
MIL-STD-2167A	DoD Standard 2167A, "Defense System Software Development"
ModSAF	Modular SAFOR
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OPFOR	Opposing Forces
OSA	Office of the Secretary of the Army
PIDS	Prime Item Development Specification
PDU	Protocol Data Unit
PM	Program Manager
POP	Persistent Object Protocol
R&D	Research and Development

S&T	Science and Technology
SAC	Senate Appropriations Committee
SAFOR	Semi-Automated Forces
SAIC	Science Applications International Corporation
SEOD	SAFOR Entity Object Database
SIMNET	Simulator Networking
SLOC	Separate Lines of Code
SRS	Software Requirements Specification
STRICOM	Simulation, Training, and Instrumentation Command
STOW	Synthetic Theater of War
SUMEX	Summer Exercise
T&E	Test and Evaluation
UCI	User-Computer Interface
USATRADOC	Army Training and Doctrine Command
VVA	Verification, Validation, and Accreditation
WARSIM	Warfighter's Simulation
WISSARD	What-If Simulation System for Advanced Research and Development

OVERVIEW

A. BACKGROUND

Semi-automated forces (SAFOR) are a key component of the Distributed Interactive Simulation (DIS) virtual environment. As the use of DIS has increased, so also has the number of programs requiring high quality SAFOR. Program officers and research and development sponsors have responded to this demand by funding independent SAFOR developments, each based on their separate assessments of needs and perceptions of the electronic battlefield. The DoD environment that evolved during the Cold War era allowed for, and even encouraged, these independent developments. However, the post Cold War environment of declining budgets requires different strategies. The exchange of ideas and products to promote efficient development, lower costs, and transfer of knowledge among interdependent programs is now at a premium.

Two major SAFOR development programs are ModSAF and Close Combat Tactical Trainer (CCTT) SAFOR. ModSAF provides an open, modular architecture that allows users to develop their own SAFOR entities and exchange them with developers and users in other programs. ModSAF is proving to be an excellent product, but it was intended more for a research and development environment than for production environment. CCTT SAFOR is being developed as a product under the CCTT production program, which is part of the larger Combined Arms Tactical Trainer (CATT) effort. CCTT SAFOR is intended to be the core SAFOR for follow-on CATT development and a source of transportable modules that can be ported to other programs so that they can take advantage of the CATT investment.

Development of SAFOR is not limited to the ModSAF and CCTT programs, nor is it limited to training applications. Both SAFOR and DIS are expected to serve many communities, including those that support acquisition, test and evaluation, tactical doctrine development, and various user communities. The exchange of ideas and products among these programs is a core issue for all concerned with the development and application of DIS.

To address this issue and help ensure coordination and cooperation among all lines of SAFOR development, the Program Manager, CATT, convened a panel to consider the following questions:

1 Can the same SAFOR products (including specifications, functional description, and software) support both the research and development community and the user community (including the education, training, doctrine, analysis, test and evaluation, production, and logistics communities)?

If yes, what are the products, what attributes should they possess, and how should they be exchanged among users?

If no, are there sub-products or components that can be used by both the R&D and user communities? If there are, what are these components and/or their characteristics?

2. What software approach (language, system, shells, tools, etc.) should be used for CCTT SAFOR considering the needs of both the research and development community (flexibility, ease of use, etc.) and user communities (controlled configuration, life cycle maintainability, reliability, etc.)?

3. What products are available from other programs, such as ModSAF and the Institute for Simulation and Training's Computer Generated Forces (IST CGF), that can be used directly or in some re-engineered form to aid in the development of the CCTT SAFOR?

4. What strategy or specific steps should the Program Manager, CATT, take to build community consensus and to produce a product that will support CATT programs other than CCTT?

5. Taking into consideration all the above, what strategy or specific steps should the Program Manager, CATT, take to implement the CCTT SAFOR?

The panel was asked to address these questions specifically, but its discussions were not limited to them. Basically, the discussions were intended to provide candid, technical interchange among groups concerned with development of SAFOR capabilities for CCTT and CATT.

The discussions were held 27-29 October 1993, in Orlando, Florida, in accord with the following agenda:

Wednesday, 27 October

0800-1000 Introduction and Discussions with PM CATT

1000-1030 Break

1030-1200 Discussion of DMSO SAFOR Survey (Organized by MITRE Corporation)

1200-1300 Lunch

1300-1500 Discussion of ModSAF (Organized by Loral Corporation)

1500-1530 Break

1530-1730 Computer Generated Forces and C to Ada conversions (Organized by the Institute for Simulation and Training, University of Central Florida)

Thursday, 28 October

0800-0900 Plans for future SAFOR capabilities and development (Presented by ARPA)

0900-1200 Current CCTT status and plans (Organized by IBM Corporation)

1200-1300 Lunch

1300-1530 Continued discussions of CCTT status and plans

1530-1730 Discussions among panel members and clarifications of information presented earlier

Friday, 29 October

0800-1100 Discussions among panel members and preparation of recommendations

1100-1200 Debrief to Program Manager, CATT

The discussions were open during the information briefings (i.e., from 1030 on Wednesday to 1530 on Thursday). Other times were reserved for the panel. Hard copies of the slides used for the presentations are included as Appendices to this document.

The six members of the panel were:

Dr. Philip Anton
MITRE Corporation

Dr. Peter Brooks
Institute for Defense Analyses

General Paul Gorman (USA, Ret.)
Cardinal Point, Inc.

Dr. John Laird
University of Michigan

Dr. Duncan Miller
MIT Lincoln Laboratory

LTC Robert Richbourg, USA
U.S. Military Academy, West Point

Prior to the meeting, the panel members received a read-ahead package that contained:

- Brooks, R.A., Buchanan, B.G., Lenat, D.B., McKeown, D.M., and Fletcher, J.D. *Panel Review of the Semi-Automated Forces* (IDA Document D-661). Alexandria, VA: Institute for Defense Analyses, September 1989—Summary Only.
- Booker, L., Brooks, P., Garrett, R., Giddings, V., Salisbury, M., and Worley, R. *1993 DMSO Survey of Semi-Automated Forces*. Washington, DC: Department of Defense Modeling and Simulation Office, 30 July 1993.
- *Report of the Defense Science Board Task Force on Simulation, Readiness, and Prototyping*. Washington, DC: Office of the Under Secretary of Defense for Acquisition, January 1993.
- *SAFOR Trade Study Results*. Orlando, FL: Project Manager Combined Arms Tactical Trainer, U.S. Army Simulation, Training, and Instrumentation Command, 15 March 1993.
- *A Modular Solution for Semi-Automated Forces: ModSAF, An Overview*. Cambridge, MA: Loral Advanced Distributed Simulation (Undated)—Briefing (Paper Copy of Overhead Transparencies).
- *PIDS Revision A: Requirements Allocated to All or SAF Computer Software Configuration Items (CSCI)*. Orlando, FL: Project Manager Combined Arms Tactical Trainer, U.S. Army Simulation, Training, and Instrumentation Command, 26 August 1993.

Members of the panel were requested to document their impressions, suggestions, and recommendations. Their comments are included as Appendix A. A summary of their comments follows. It is divided into three sections: Findings, Recommendations, and Conclusions.

B. FINDINGS

Presently the CCTT SAFOR has been designed as a re-engineered version of ModSAF to be developed using an Ada programming environment. ModSAF itself and

nearly all SAFOR applications have been developed using C programming environments. A major decision for PM CATT is whether to pursue an Ada-based CCTT SAFOR or to require more compatibility with the existing C-based development of ModSAF. This decision impacts the development of SAFOR capabilities across all applications and concerns all questions raised by PM CATT for this panel.

The following discussion summarizes the findings of the panel:

1. Immaturity of SAFOR Technology

All of the panelists emphasized that SAFOR is rapidly changing and evolving—that it is not a mature technology ready for "type classification" and fielding. CCTT SAFOR will remain a dynamic area for several years to come, and ModSAF itself needs continued development. Even if designs and approaches for SAFOR were more settled and understood, the new post-Cold War threat environment, which is equivalently dynamic and rapidly evolving, will demand quick and occasionally urgent representation of new opponents, allies, terrain, and situations in SAFOR. These representations may become available from other programs entirely separate from SAFOR and DIS. The degree to which CCTT and CATT SAFOR developments remain flexible and capable of easily incorporating—without major re-engineering or re-design—useful products and ideas from all these rapidly evolving sources will substantially impact the quality, relevance, and utility of their products.

2. Coordination with Research and Development

All of the panelists discussed the specific need to coordinate research and development products with those produced for CATT SAFOR. They raised the following points:

(a) CATT SAFOR development will help focus research priorities, frame research questions, and provide a meaningful baseline with which to compare new research results. However, CCTT SAFOR is an engineering development program that needs an externally provided science and technology base. The flow of products and ideas from research and development to CATT, as discussed above, is important. Also important is the flow of ideas and products in the opposite direction, from CATT to research and development efforts. These efforts will need access to genuine, user-produced SAFOR modules. Employment of CATT products for this purpose will substantially improve the quality and relevance of products from research and development efforts, and it will improve the efficiency with which they are produced.

(b) It is unlikely that CCTT and CATT modules will be used as directly substitutable "black-boxes" by researchers. Even if standardized interfaces can be designed, constructed, and enforced to meet SAFOR needs for product exchange, researchers will still need to make modifications within modules to accommodate their objectives. A mixed bag of modules in various languages will discourage these modifications and their potential for promoting reuse and coordination even if the interface specifications are well defined. These modifications will be practicable for researchers only if they can be accomplished using software tools, architectures, and approaches with which they are familiar.

(c) The panelists noted specifically that the planned development of 1200 Combat Instruction Sets (CIS) will be "the largest representation of intelligent human behavior ever undertaken" and a major step forward in representing military behavior. No other program will have the resources in the foreseeable future needed to create or re-create this body of knowledge. It is therefore essential for the CIS modules to be easily available not only to all members of the research and development community, but to all developers concerned with DIS and otherwise. Few of these developers and fewer researchers will be able to find the resources needed to adopt CIS modules from one programming architecture and environment to another.

(d) Without coordination between these communities, discrepancies between their products, functionalities, approaches, and designs will inevitably arise, testing and calibration, which is difficult enough in the SAFOR environment, will become more complicated, and products from the technology base will be harder to produce, less relevant, and more difficult to incorporate. If this coordination is achieved, shared software expertise and rapid exchange of products and ideas will increase the efficiency with which all lines of SAFOR development proceed, and it will increase the quality of their products.

3. Breadth of the SAFOR Requirement

Five panelists emphasized that the issues raised by CATT SAFOR development are broader than requirements for CATT alone—significant as that program is. The community of SAFOR users is larger and more diverse than that originally contemplated and it continues to grow. For instance, Lt. General Forster, the Deputy Acquisition Executive of the Army, has recently directed Army acquisition executives to use simulation for all Army acquisition programs, thereby further increasing the body of SAFOR developers and users. To be accepted in these communities, CATT SAFOR must provide validated data and models, operator suitability, a ready ability to test and calibrate the system, and use in applications beyond training. Success for CATT SAFOR development

may be measured as much by wide user acceptance and its support for the full community of SAFOR users as by its support for development within the CATT family of systems.

4. Re-engineering of ModSAF

Two panelists discussed their favorable impressions of both the ModSAF architecture and its re-engineering for CCTT. They suggested that the CCTT architecture is a commendable improvement. On the other hand, they pointed out that the re-engineered architecture is itself untested and unproved—certainly less so than ModSAF. The re-engineered CCTT SAFOR architecture should be as subject to careful scrutiny as any other SAFOR approach.

5. Quality of Software Engineering

Three panelists emphasized that it is the programmer more than the language that produces well-engineered software, allowing efficient maintenance and reducing life-cycle resource requirements. Most specifically, the software engineering discipline imposed by an Ada programming environment can be also be enforced by programmers using the C and/or C++ environments that are used for SAFOR developments elsewhere.

6. Commercial Tools

Four panelists noted that the accessibility of any software product will be enhanced by the use of tools that are available at reasonable cost and that operate on a variety of computing platforms. These observations arose directly from the proposed used of RTWorks in plans for CCTT SAFOR, but they generalize to the use of any tool for software development or operation.

7. National Guard Requirements

One panelist discussed the emerging and increased responsibilities of the National Guard in readiness and direct combat preparedness. These responsibilities suggest increased needs to enhance small unit performance and support mission rehearsal. Significant steps toward meeting these needs are provided by the editor functions in ModSAF and the production of modules that are compatible with these editors. The National Guard's need for behavioral detail at the small unit level should not be lost to the demands of active component commanders for greater levels of aggregation.

8. C and C++ Programming Environments

Two panelists discussed the extent of resources available in C and C++ programming environments. More specifically they argued that C and C++ environments provide all the basics of good software engineering and practice such as systems, tools, compiler speed, object orientation, efficient code generation, concurrency support, hierarchical capabilities, abstraction, encapsulation, modularity, and even strong typing. They also argued that these resources are more readily available from C and C++ than from Ada environments, due in significant measure to the greater number of C programmers, systems, and tools, and lower costs in terms of time, budget, and computer resources.

C. RECOMMENDATIONS

1. All panel members recommended that CCTT/CATT SAFOR should be developed to facilitate and maximize interchange between its products and ideas and those of all other communities, but especially those of the R&D community. Development of parallel but computationally different SAFOR systems should not be pursued nor supported.
2. All panel members concluded that CCTT SAFOR should not be developed in Ada. It should be developed in C or in C++. The costs—in terms of resources and time—of this recommendation were taken into consideration. All members of the panel concluded that benefits arising from this decision outweigh its cost.
3. Four panel members emphasized that products from the CATT SAFOR developments should be made as accessible as possible and that the use of elaborate commercial shells and tools should be avoided.
4. Four panel members specifically recommended the use of the C Language Integrated Production System (CLIPS) in place of RTWorks. One member further recommended the use of CLIPS Object-Oriented Language (COOL).
5. Two panel members specifically recommended the use of object-oriented analysis and design tools to provide traceability and design documentation.
6. One panel member recommended appointment of an advisory panel with representatives from all concerned SAFOR communities to provide continued assistance to PM CATT. More generally, another member recommended that PM CATT continue to solicit review and feedback from the research and development community.
7. Three panel members commended the use of people in addition to careful documentation and good software engineering practice to effect coordination

between CCTT SAFOR production and other communities, and they recommended that this practice be continued.

8. One panel member commended the emphasis on human computer interaction in CCTT SAFOR development and recommended that it receive continued emphasis.
9. Three panel members commended the emphases on concurrent engineering and cyclic simulation feedback for CIS development and recommended that these approaches continue to receive emphasis.
10. Five panel members recommended a strong insistence on good software engineering practices. PM CATT should not rely on the enforcement capabilities inherent in a programming language. Module interface design, documentation, data abstraction (with data separated from code), and standardized implementation via MIL-STD-2167A should all receive continued emphasis.
11. Three panel members recommended that flexibility, accessibility, and adaptability be weighed more than life-cycle maintainability in the development of CATT SAFOR products.
12. One panel member recommended that priority in SAFOR development be given to entity-level behavior and small unit performance.
13. Two panel members specifically commended the Integrated Development Team's ModSAF re-engineering effort, and they recommended that further development build on this effort where possible.
14. One panel member recommended that, if development continues in Ada, at least one additional round of SAFOR enhancements should be undertaken before the system is fielded.
15. Two panel members recommended that the CATT development community should take a more active role in funding research—specifically it should pick up what ARPA may leave unaddressed or unfinished.

D. CONCLUSIONS

In brief, the panel concluded that the widest possible community should develop and share ownership in a CCTT SAFOR product that is produced using the best available notions for software engineering. More specifically:

- The same SAFOR products can and should be used to support both the research and development and the user community;

- Two computationally separate SAFOR lines of development, one based on an Ada environment and one based on a C environment, would inevitably develop discrepancies, become insufficiently coordinated, and should not be pursued;
- The research and development community and other interested communities are unlikely to have either the resources or inclination to migrate to an Ada programming environment;
- Products from CATT SAFOR development should be made as accessible and adaptable as possible—higher priority should be given to accessibility and adaptability than to life-cycle maintainability;
- CATT SAFOR development should be pursued using a C and/or C++ programming environment.

APPENDIX A

PANEL MEMBERS' COMMENTS

COMMENTS BY PHILIP ANTON

1. SAFOR PRODUCTS

SAFOR is an immature field with rapidly evolving techniques, components, and requirements. As such, it is very difficult but not impossible to construct SAFOR products that meet some of the common needs of the R&D and User communities. Such products must, however, possess several properties, including flexibility, rigorously defined interfaces, and design with "shared" requirements in mind.

As components become stabilized and approaches become common, those components can be reused by the SAFOR community at large. Note, however, that standardization does not imply restricted access to components. The R&D community has the task of continually investigating SAFOR architectures as a whole and will continue to need the flexibility to investigate modifications or alternative approaches of components. DIS PDU standards may be somewhat stable in their description of entity movement, but solutions to the problems of scalability and fault tolerance may require modifications not of the network itself but of the entire approach to SAFOR architectures to include entity and behavioral representations, PDU generation approaches, and database structures. The R&D community must have the ability to test out alternative solutions to such problems if the state of the art is to be advanced.

If we are to reuse components of the architecture, then the interfaces between the components must be rigorously defined to meet common requirements.

Common products that should be sharable today include:

- A generic SAFOR architectural shell, including
  - -- DIS PDU interfaces
  - -- simulation support, including stochastic and deterministic options
  - -- dead reckoning algorithms
  - -- coordinate system transformation algorithms
  - -- libraries of behavioral processing engines, including task frames, rule-based inference engines, etc.

- Knowledge bases of entity behaviors (e.g., Combat Instruction Sets, or CIS)
- Databases of vehicular dynamic parameters.

Components that should NOT be forced into a baseline SAFOR include:

- Proprietary/COTS tools
  - cost barriers to use
  - inaccessibility for study and extension by the research community
  - dependence on the supplying company for all future developments.

Government or research developed software may cost more initially to develop, but in the long run it can reduce costs and increase longevity if proper care to support reuse and common design is employed.

Note that the knowledge bases and databases should be constructed in an architecture-independent fashion whenever possible. Knowledge bases implemented as data rather than in source code languages (e.g., Ada or C code) allow for continued development of the processors of the knowledge data.

The degree to which the same SAFOR products can support different communities is a function of

- the degree to which common requirements can be found
- the stability of the design and implementation of components which can be reflected either in the formal establishment or the SAFOR community.

Even within a single community, there will be a core of needs based on the types of processing that community does, but studies will often need to be performed beyond the current capabilities of the community's systems. Most people need a module that handles the standard DIS PDUs, but no one yet agrees on the approaches that should be used in behavioral representation. The OSA DISC4 ASRMO briefing distributed during the panel emphasized this stability requirement in the reuse of software.

Standards, of course, are problematic in and of themselves. Design-by-committee is known to result in standards that at best meet some of the requirements across the communities and at worst do not meet anyone's requirements due to extensive compromise. The research community by its nature needs the flexibility to continue to study standardized components for improvement.

Functional interfaces to allow experimentation, replacement, and remixing of modules hold the greatest immediate promise in establishing specifications that can be

reused. Care must be taken, however, to design these interfaces in a flexible and open architecture to maximize the ability to replace and upgrade modules as time goes on while providing a common environment.

There are a number of reasons why it is imperative today to coordinate SAFOR development in the R&D and User communities. First, dwindling DoD funding implies that custom software development for each developed system can no longer continue. Reduced funding also implies that each system development project cannot afford its own Science and Technology (S&T) research programs to support its unique set of requirements. Reduced funding also means that less research dollars will be available and more research will have to be directed to meet the immediate needs of the user community. Thus, the R&D and User communities must establish ways to work together to meet common requirements in addition to meeting their own unique requirements. This will require coordinated efforts and discussions on technological approaches to problems as well as the establishment of methods to support the direct application of research results into development systems as well as the availability and use of validated user systems in research programs. Not only will the feedback of systems from the user community to the R&D community save development money of common modules, but research in the user's environment will focus researchers on user problems rather than abstract technological problems.

There is an unexploited opportunity in the SAFOR community to reuse not only SAFOR software but the architectures, specifications, and functional descriptions (e.g., component techniques, algorithms). If common requirements can be found between R&D and development programs, then the effort of the receiver of products and software should concentrate their effort not on re-engineering the received software but on producing the 2167A documentation of the existing software. For example, ARPA and CCTT have invested significant funds to develop ModSAF. Components that met the CCTT requirements (e.g., PDU handler) could have been used directly by reverse engineering appropriate IDS, IDD, and SRS documents to describe this module. These specifications and functional descriptions would then be available for other users of ModSAF, resulting in significant reuse and cost savings. If one program upgrades the PDU handler and associated documentation to the next version of the DIS standards, then other users (including the R&D community) could immediately upgrade to the new standard with little or no effort. If, however, the PDU handler is re-engineered into a different language, then other users of ModSAF will not be able to use it. Having a mixed bag of modules in

various languages will not promote reuse of the modules even if the interface specifications are well defined.

If a developer needs to extend a module's capability to meet a specific program requirement, then that module could be customized to meet these new capabilities. The issue here is whether these extensions meet the needs of the broad SAFOR community or just the present program under development. If consensus is reached in the community at large that the extensions are useful and should be incorporated in the baseline SAFOR, the module could readily be incorporated in everyone's SAFOR easily if the same implementation languages is used. If, however, the extensions are not needed or agreed upon by the community, then the module should not be forced upon the community. For example, if CCTT SAFOR is developed in Ada or is a complete re-engineering of ModSAF (as is currently planned) and the extensions that the IDT have included in the SAFOR architecture do not meet a consensus requirement by the community (e.g., if the SAF Entity Object Database - SEOD - extensions of the Persistent Object Protocol - POP - are not what the SAFOR community at large needs), then providing CCTT SAFOR back to the R&D and user communities will "force" these non-consensus designs on the community or limit the reusability of CCTT as a baseline. Only if CCTT is developed in the common language of the SAFOR community and the extensions provided by individuals are discardable or usable by virtue of well-defined interfaces can the reuse of SAFOR products provide valuable growth of a baseline SAFOR environment.

Note that the CCTT design presented at the workshop is "not" a mere re-engineering and development of existing SAFOR ideas and research results. The CCTT SAFOR design itself is a new, unproven architecture and thus constitutes a research development effort. Who is to say that the architecture is sufficient to meet the stated goals of CCTT and latter CATT? If ModSAF was insufficient, then a re-engineering of ModSAF (with small changes and the inclusion of a rule-based shell) may be insufficient also. But even then, ModSAF has yet to be delivered and demonstrated in a real, validated exercise.

2. SOFTWARE APPROACHES

In order to satisfy both the R&D and user communities, CCTT SAFOR should use a computer language that is efficient, standardized, in common use in both communities, allows good software engineering practices, and has compilers that are fast, readily available, and low cost. The obvious candidates are Ada, C, and C++.

Ada meets most of the requirements but its compilers are slow, expensive, and not commonly used in the R&D community. A good Ada compiler can provide reasonably efficient code and provides tight error checking in support of software development, but the lack of Ada availability in the R&D community would prevent the use of CCTT SAFOR as a research baseline for SAFOR studies.

C and C++ are very efficient, standardized languages used extensively in both communities. GNU compilers for C and C++ (published by the Free Software Foundation) are free, readily available for common hardware platforms, and among the highest quality available, often surpassing the compilers delivered by hardware vendors. These compilers are heavily used by the R&D community as well as commercial software developers. While C (and to a lesser extent C++) do not provide as much software engineering support, the object-oriented analysis (OOA) and design (OOD) approaches already adopted by the CCTT IDT will provide significant software engineering support and traceability from specifications to code. In addition, the quality of the code developed ultimately depends on the quality of the programmer, not on the language chosen. Good programmers can produce well-designed, quality code in any language, and poor programmers can produce poor code in any language (including Ada).

Judicious use of C++'s object-oriented features could provide flexible object implementations while minimizing inefficiencies inherent in object-oriented languages, but care must be taken. The ModSAF team made a deliberate (and allegedly informed) design decision to provide a custom object-based (not object-oriented) implementation of entity components for efficiency reasons. Creation of complex inheritance structures of objects combined with uninformed use of the language can result in unexpected inefficiencies. Nevertheless, C++ does provide fundamental object-oriented features that can facilitate flexible design in an efficient manner.

Note that question 2 poses that the R&D needs of flexibility and ease-of-use are pitted against the user community needs of configuration control, life-cycle maintenance, and reliability issues. In the immature field of SAFOR, however, things are not this clear cut. CCTT will need to continue to fold in new research results and approaches (e.g., C2 Simulation Interface Language—CCSIL, results in Behavioral Representation and Dynamic Terrain) from the research community as they are proven and become available. Thus, CCTT will not have a traditional life cycle of static requirements, implementation, and slow software maintenance. CCTT itself must remain flexible, open, and easy to use in order to take advantage of breakthroughs in SAFOR technology and remain useful.

Thus, overemphasis on traditional life-cycle maintenance (for which Ada claims excellence) at the expense of flexibility does not match the CCTT (and CATT) role in Army training. CCTT and CATT has (or should have) the explicit requirement of including the needs of the R&D community since it is a critical part of the "life-cycle" of the CATT program.

As for the use of software shells, I would like to argue strongly against the use of commercial shells. The use of shells such as RTWorks may provide extensive support facilities for CCTT development, but the inclusion of such commercial products will greatly limit who will be able to use the CCTT SAFOR environment, including the very research organizations to whom CATT will turn for results to meet the CATT goals. Also, the SAFOR community is still struggling with what types of behavioral representations are appropriate for what types and levels of battlefield entities. A rule-based system may be useful for higher-level aggregate entities, but there has been no research on this point to date and certainly no data to demonstrate that the syntax provided in RTWorks is necessary and sufficient to express the behavioral rules for Army units.

There are many well-engineered inference engine shells. CLIPS is a GOTS, C-based, real-time inference engine shell that supports rule-based, object-oriented, and procedural paradigms and comes with X-based development tools. It is highly recommended in the research community and commonly used. CLIPS is available from the Software Technology Branch of the NASA Johnson Space Center, is free to government agencies and contractors, and relatively inexpensive for others ($150–300 range).

If CCTT must use RTWorks, then a requirement should be formally placed on the IDT to specify the interface between RTWorks and the rest of CCTT SAFOR to allow easy removal or replacement of the RTWorks inference engine for research studies and other SAFOR developments. Knowledge bases (i.e., CIS data) should employ the rule-based system in such as way to allow the rule-based engine to be functionally replaceable with a different behavioral system (e.g., a different knowledge-based system, SOAR, planner, probabilistic reasoning, etc.).

Thus, I recommend that CCTT SAFOR be implemented in C or C++ (with preference to C++). Object-oriented analysis and design tools should continue to be employed for CCTT; these tools provide the critical requirements traceability and design documentation for CCTT development under 2167A while providing important documentation for other users of CCTT SAFOR code.

3. OTHER AVAILABLE PRODUCTS

ModSAF ideas have already been re-engineered into the CCTT SAF, and I have no direct experience with IST CGF. I would have recommended a more direct use of ModSAF code with an associated effort to reverse engineer just the 2167A documents for the ModSAF architecture. Unfortunately, the re-engineering decision has already been made.

As for other tools that could be used to aid the development of CCTT SAFOR, I would strongly recommend the use of the GOTS CLIPS inference system rather than RTWorks. If CLIPS were to be used, I would recommend reverse engineering of the 2167A documents for CLIPS (if not already available) rather than a complete re-engineering effort. This would allow other users to have easy access to the inference engine of CCTT at a negligible cost, save re-engineering costs for CCTT, and allow CCTT to give back to the R&D community the specifications for CLIPS so that other programs that need such an engine would be able to re-use these specifications.

4. COMMUNITY CONSENSUS STEPS

Consideration of R&D community requirements when making programmatic and design decisions is crucial to supporting CCTT and other CATT programs. If CATT is to succeed, then it must be flexible enough to include research results in SAFOR. Also, given the limited R&D funds, CATT must support the R&D community by promoting a SAFOR environment to focus research on user problems and provide a validated environment in which to perform the research.

To promote consensus, PM-CATT should convene an advisory panel to include representatives from both the R&D and user communities (possibly including joint service representatives) to make continued recommendations on design decisions. Difficult questions need to be addressed and recommendations made as a result of the IDT decisions to date:

(i) If CATT is developed in C or C++, how can the ongoing ModSAF development be integrated with the CATT delivery system? What parts of the CATT system can be used by a generic SAFOR and thus included in a re-merged ModSAF-CCTT system?

(ii) Review and make recommendations on the interface specifications between the modules in the CATT architecture. Should an effort be started to define generic open-

architecture interface standards to facilitate SAFOR "Plug and play" of components? Can the WARSIM 2000 architecture be used for this? What progress has WARSIM made and can the CATT interfaces be specified within the WARSIM architecture to facilitate coordination with WARSIM in the future?

(iii) Review and make recommendations on the behavioral representation design decisions in CATT, to include:

- the interface between RTWorks and the rest of the system,
- the design decisions regarding the implementation of the CIS database.

For example, will CISs be implemented as source code (as in the ModSAF tasks) or can they be implemented as data (e.g., ASCI text) in a certain syntax to be operated on by processing engine(s) in the architecture, independent of the engine?

(iv) How can the current ARPA research efforts (e.g., CCSIL) be designed and developed to facilitate direct use in CATT programs as needed without a re-engineering effort? Who should fund and develop IDS and SRS for such software modules?

(v) If continued coordination is necessary between the R&D and user communities for SAFOR, what arrangements and structures can be established to facilitate this coordination? Should ARPA be the keeper of a baseline SAFOR environment? If not, then who? DMSO? The Army?

(vi) Given the shrinking DoD funds for developing new systems, what advice can be given to future Program Managers on how to control contractor's natural tendency to want to build custom systems rather than reuse existing software whenever possible?

(vii) If the Army becomes the repository of the baseline SAFOR, will other services be less inclined to reuse it than if the software came from a service-independent source (e.g., ARPA or DMSO)? How can such "rice bowl" issues be reduced in a joint environment?

(viii) CCTT has demonstrated that personal interaction between software developers (e.g., LADS) and the software re-users (e.g., CCTT IDT) was very valuable in transferring an understanding of ModSAF. Unfortunately, this kind of personal interaction is not possible in all cases given availability and cost considerations. What types of documentation would facilitate software, architecture, and algorithm transfer between communities given the need to re-use designs? Should researchers (or someone following up on the research's work) spend the time to document the code using 2167A require-

ments? What other techniques could be employed? Note here that the emphasis should not be to manage by consensus or committee but to bring to light issues important to coordinated development and re-use of SAFOR products and to make recommendations to PM-CATT on how to meet the immediate CATT needs while considering the requirements of the SAFOR community as a whole.

5. SPECIFIC STEPS TO IMPLEMENT CCTT SAFOR

In summary, I recommend:

- Implementation of CCTT in C++ (or C).
- Continued use of OOA and OOD tools.
- Emphasize good software engineering practices rather than reliance on a software language's inherent enforcement of certain practices (e.g., Software Engineering Institute ratings).
- Continue special attention to the UCI, which is critical if the system is to be successful.
- Continued emphases on concurrent engineering and cyclic simulation feedback for CIS development.
- Consider the use of nonproprietary software in place of RTWorks (e.g., CLIPS).
- Consider the CIS implementation format and the impact of this design decision on reusability of the CIS data (i.e., implement each CIS as data versus software code). Can a generic description of CIS components implemented as rules be reached independent of the syntax of the CCTT inference engine?
- Pay special attention to explicit and rigorous interface design, documentation, and implementation for the CCTT architecture modules to facilitate re-use and individual replacement for research and development in other SAFOR systems.
- Negotiate a plan for integrating CCTT with ModSAF, replacing parts of ModSAF with parts of CCTT, or some other approach to provide some kind of common SAFOR baseline environment that other SAFOR programs can build on.
- Seek the advice of both the R&D and user communities in addressing the questions outlined above in section 4.

The major design concerns identified are:

- The impact of the use of the SEOD on scalability in CATT SAFOR (an open research issue).

- The inclusion of proprietary software in the CCTT architecture and the degree to which this software is integral to the whole CCTT system.

Lessons Learned:

- Concurrent validation with knowledge engineering efforts should be promoted. Similarly, iterative simulation feedback should be employed in addition to extraction of textual description from Subject Matter Experts and doctrinal documents (ref. CIS and WISSARD).
- Greater care must be paid to explicitly issuing related requirements during initial program studies if any other community's needs are a factor in programmatic decisions. For example, the requirement of flexibility in rapidly incorporating research results in CATT programs as well as the need to offer CCTT SAFOR to other programs for research and development purposes needed to be included as explicit criteria in the CCTT IDT SAFOR Trade Study. As the R&D and user communities need to rely on each other and leverage each other's work, this reliance needs to formally be recognized in program requirements. Note that developers and managers have a natural tendency to make design decisions to maximize the perceived risk to the explicit requirements even if these decisions are not the best compromise for the explicit and implicit requirements as a whole.
- The use of independent panels such as ours (hopefully) helps to bring fresh perspectives to bear on programmatic issues as well as support cross fertilizing of information between communities regarding important programs that will impact them.
- "Optimize for change, since change will surely come." (Gen. Paul Gorman)

COMMENTS BY PETER BROOKS

SUMMARY

The Program Manager, CATT, asked the Panel to consider CCTT SAFOR from two perspectives. First, is the current approach for CCTT SAFOR well-suited to the needs of other SAFOR development and user communities? Second, what products and strategies would enhance the development of CCTT SAFOR?

Several key observations emerged during the briefings and subsequent discussion:

- It is important for CCTT to demonstrate that a common development environment exists for DIS applications.
- SAFOR technology will remain an active research and development area for several years.
- CCTT is a 6.4 program, and thus needs an external science and technology base.
- CCTT SAFORs will be successful only if it attains wide user acceptance. Key factors include validated data and models, operator suitability, a ready ability to test and calibrate the SAFOR system, and use in applications beyond training (e.g., acquisition support).

Based on the discussions, two potential recommendations are:

- PM CATT should establish programmatic ties and maintain commonality with the ARPA SAFOR research efforts.
- PM CATT should ensure that the CCTT SAFOR is readily accessible to broad user and developer communities.

There are several elements of the current development strategy worth noting. These include:

- the high degree of user and proponent involvement.
- the structure of and process for developing the Combat Instruction Sets.
- the relocation of personnel among the IDT and Loral sites.
- the study of how ModSAF might be re-engineered.
- the efforts to maintain traceability from requirements through development.

The above observations argue for using software tools, techniques, and languages favored by the R&D community.

THE BENEFITS OF A COMMON DEVELOPMENT ENVIRONMENT

CCTT, as the first major program in Distributed Interactive Simulation (DIS) since the conclusion of SIMNET, represents a key test case for the development and use of DIS. The premise of DIS today is that each development program adds to the whole, that new capabilities are easily added into existing systems, and that DIS is generally useful for applications beyond training.

CCTT must thus demonstrate that major software components and development techniques can be re-used. The benefits are clear in the context of the other CATT programs, and one would expect a high level of commonality from the start. But the development also should contribute to a Joint DIS-based training system. In this vein, there seems to be too little consideration to how CCTT will operate on a wide area network (CCTT has high bandwidth requirements; local net is FDDI), though this is identified as a preplanned product improvement.

For user communities beyond training (e.g., acquisition), there will be the need to add new capabilities or modules to CCTT SAFOR, and to exercise SAFOR in their own laboratories. These users must be able to replicate and easily work in the development environment established for CCTT SAFOR.

The "entry cost" to CCTT SAFOR therefore must remain low, through the use of commonly used hardware and software (e.g., C or C++), and the lack of proprietary or expensive components (e.g., will RTWorks make CCTT SAFOR unaffordable to most researchers?).

SAFOR TECHNOLOGY IS AN ACTIVE R&D AREA

The ARPA research in SAFOR will continue for several more years, at a funding level several times that of CCTT SAFOR. CCTT should provide for the easy incorporation of new SAFOR technologies by ARPA.

SAFOR will evolve for other reasons as well. New users will demand or develop added capabilities (cf. Gen. Forster directive to Army Acquisition Executives to use simulation for all systems acquisition programs). In other cases, new SAFOR capabilities

will be added due to pressing needs (e.g., SAFOR vehicles being given a newly developed vehicle mounted countermine system).

These points argue for having the CCTT SAFOR system and the baseline research SAFOR system be largely interchangeable.

CCTT REQUIRES AN EXTERNAL SCIENCE AND TECHNOLOGY BASE

Because CCTT uses 6.4 funding, it relies on external programs for research and development work. Many organizations will contribute to improving SAFOR (e.g., terrain reasoning, command and control algorithms, new munitions effects, programs for the individual soldier, etc.). To incorporate these contributions may require more than well-defined interfaces that treat SAFOR as a black box. Such general interfaces may be too hard too construct, anyway. Instead, it may be necessary for the researchers to work with the code directly.

If CCTT is relying on ARPA as the main source of research directed at SAFOR, then such advances should be incorporated into CCTT without extensive re-engineering of each added capability. A programmatic connection between CCTT and ARPA may help here.

CCTT SAFORS SUCCESS BASED ON WIDE USER ACCEPTANCE

CCTT SAFOR will be viewed as the official SAFOR, and therefore the system of choice, as it will contain Army-validated behaviors and data. To ensure its utility to user communities beyond training will require better performance in the user-computer interface (UCI) and in SAFOR testing than exist today.

The human factors analyses of the usability of the UCI should consider how SAFOR is used in various applications. For example, does the IDA analyst want to finely control the movements of one or two helicopters? Does the AI researcher need real-time explanations of the internal decisions SAFOR is making? Will there be SUMEX-like test for other user groups?

Testing and calibrating SAFOR behaviors will always be difficult, mainly because it runs only in real-time (even if the computer is powerful to speed up a given scenario, there is no guarantee that the approximations made in vehicle movements might not change the results). Techniques should be developed that assist the full-up system testing of SAFOR.

Also, one may need a regular program of manned simulator tests to provide a basis for calibrating the behaviors encoded in the CISs.

POTENTIAL RECOMMENDATIONS

PM CATT should establish programmatic ties and maintain commonality with the ARPA SAFOR research efforts. This implies developing a shared baseline system, using a common programming language, and establishing a program to educate the respective developers. Developing a good SAFOR system remains a learned art.

PM CATT should ensure that the CCTT SAFOR is readily accessible to broad user and developer communities. The key factor is hardware and software cost, which can be reduced by minimizing the need for expensive compilers, commercial software, or programming expertise that would have to be specially hired.

COMMENTS ON THE CURRENT DEVELOPMENT STRATEGY

The most noteworthy elements of the development strategy are the efforts to involve proponents and users. Clearly, CCTT must end up with validated models and data. The program has involved the appropriate organizations early and fully. SUMEX '93 is a good way to educate the code developers. Such exercises should be held several times a year.

The encoding of doctrine into the Combat Instruction Sets (CISs) represents perhaps the most profound element of the program. If successful, the CISs will become ***the*** representation of doctrine, and the means by which new doctrine will be developed. The risks are that the current approach will capture what experts say they would do, and are not based on experiments using manned simulators. Should the CISs reflect how people fight, or should they represent a standard to train against?

The exchange of personnel among the IDT and Loral sites was clearly beneficial. It underlines the need for the CCTT program to have a close and continuing link to the various research efforts and expertise.

The study of how ModSAF would be re-engineered is valuable for two reasons. First, it provided an independent assessment of the good ideas in ModSAF. Second, any such reexamination is likely to make for an improved product, no matter how much or little of ModSAF is reworked.

Being able to maintain traceability from requirements through development is a good way to get outside people involved, and to manage expectations. The panel

discussions would have been enhanced by more detail of what are the CCTT SAFOR requirements.

COMMENTS ON THE PRESENTATIONS

The presentations were uniformly informative, and the presenters generally well-prepared. Much time was spent on areas with minor bearing on the key issues. The IST analysis of C to Ada conversion had too many caveats to be pertinent, for example. The IDT presentation discussed the Trade Study at length, only to later observe that it is now overtaken by events. Also, the IDT presentation on their re-engineered design of ModSAF could have discussed more the CCTT requirements which necessitated the redesign. In particular, how much of the redesign effort was due to an assumption that Ada would be used?

COMMENTS BY PAUL GORMAN

The panel was convened to inquire into the cogency of developing computer software written in the Ada computer language for the Semi-Automated Forces (SAFOR) within the U.S. Army training devices termed Close Combat Tactical Trainer (CCTT), a subset of Combined Arms Tactical Trainer (CATT). The following remarks urge that PM CATT reconsider use of Ada on the grounds that, at a time when change is the order of the day, Ada code may render CCTT SAFOR and other CATT components less accessible and mutable than C or C++.

The Ada decision had been taken some time ago on the grounds that (a) Ada is mandated by the Department of Defense and the governing requirement document, (b) Ada is specifically indicated for re-engineering mature software ready for production, in that it is comparatively error-free or error-correcting, and (c) the life-time costs for maintaining Ada would be less than if written in any other language. However, that Ada decision had been conjoined with adopting some of the features of ModSAF, an ARPA-sponsored modular approach to SAFOR programming not written in Ada, so that the current CCTT program converts the best features of ModSAF to an essentially Ada architecture. CATT managers pointed out that CCTT is moving on a demanding schedule toward production and issue to the force; that they had invested a year's time and effort in production of the SAFOR software including Ada; and that scrapping Ada would entail at least six months delay in fielding CCTT.

The names of CCTT and CATT reflect their original intent as training support mechanisms. But to manage these now exclusively for training could truncate their centrality for supporting research and development, test and evaluation, and operational rehearsal. Moreover, their Army origin and focus belies their importance for joint (multiservice) applications. Rhetoric of the Defense Science Board, and of a number of past and present officials of the Department of Defense, embraces such broader missions. Further, the recent report of the Senate Appropriations Committee on the 1994 Defense Appropriations makes it evident that the SAC considers intrinsically valuable the digital battlefield created for SIMNET, the predecessor of CATT, and intends to support extension of that environment beyond the funding requested by the Administration. Narrowly

conceived requirements notwithstanding, PM CATT can realize optimal return on the CCTT/CATT investment only by insuring that his system remains accessible, capable of being extended readily to embrace joint training, and to support communities of other users well beyond those originally contemplated. Ada appears to constrain rather than enhance accessibility.

SAFOR will assuredly be important to the acceptability and usefulness of CCTT, and ultimately that of all components of CATT. However, times have changed since the requirements for CCTT/SAFOR were written. Conceived to represent the canonical opponents in a NATO-Warsaw Pact war, SAFOR must now be mutable, able to represent with dispatch and facility any force that might confront U.S. forces. Moreover, since many SAFOR applications will represent U.S. units, SAFOR platforms must be readily changeable to reflect the continual upgrade in sensors, munitions, and other capabilities contemplated in current U.S. defense policy. In short, criteria for SAFOR software ought to accord higher value to accessibility and adaptability than to life-cycle maintainability.

Last May, Lt. General Forster, the Deputy Acquisition Executive of the Army, signed a directive requiring all Program Managers to submit a simulation plan that specifically considers all three forms of simulation (subsistent or real, virtual or apparently real, and constructive or modeled). Though the Army has stated requirements for a virtual simulation for Research, Development, Test and Evaluation (RDT&E) entitled Battlefield Distributed Simulation-Developmental (BDS-D), the sole virtual simulation presently available is SIMNET, and CCTT offers the main prospect for a generic synthetic combined arms environment to support future RDT&E. SIMNET has demonstrated that a virtual simulation such as CCTT, employed with SAFOR, can establish the context for RDT&E of a new item of materiel, from establishing its military worth early in the R&D cycle through testing its performance prior to production for the force. But Program Managers are not likely to command the funds or the programming skills required for Ada code to insert their materiel into such an environment. Again, accessibility and adaptability of the CCTT code appears to be a primary measure of its usefulness.

A very current illustration of the foregoing set of issues is presented by one of the premier projects of DDR&E Thrust Panel 5, the 21st Century Land Warrior (21 CLW). 21 CLW is a program under the aegis of the Army's Natick Research and Development Center that aims at fielding an integrated set of equipment for individual combatants: powerful new weapon(s), computer-aided command and control, and battle dress enhanced for survivability. The simulation plan for 21 CLW has thus far included only constructive

models yet to be validated. Even if these models were to prove reliable, it is not clear that they could reliably portray increases in effectiveness that would accrue through use of 21 CLW, and it is virtually certain that they would shed no light on the tough questions of man-machine interface inherent in equipment that combatants wear and personally carry, or evaluate prospects of mentally overloading the dismounted combatant. SIMNET made no provision for individual combatants; CCTT presently treats dismounted infantry fire teams as a SAFOR entity, but not as individuals, and even that representation is not validated or verified. What is plainly needed is an individual portal into virtual simulation—conceivably, a new "system" for the CATT program—and instrumentation for individual participants in subsistent simulation that, taken together, will enable comprehensive simulation of 21 CLW to compare its military worth against current equipment, and to assess its implications for doctrine and force structure. Further, since modern missions for U.S. armed forces place heavy demands on dismounted troops, entity level SAFOR portrayal of dismounted combatants, friendly or enemy, would be strategically useful. I-Port does not now exist, and SAFOR for individual combatants requires research and development at the P6.1-P6.2 (Science and Technology) level. Developers of such tools will almost surely find onerous working with an Ada encoded CCTT.

In implementing CCTT SAFOR and indeed, in addressing other similarly important management issues, PM CATT must consider the intended primary users of his fielded system. CATT must enable training on a synthetic battlefield of heavy forces being readied for close combat. Most users of CATT will be National Guardsmen in company armories whose attention will center on minor tactics. For these, SAFOR can and should portray opposing forces (Red SAFOR) to establish uniform conditions and standards for tasks assigned during well-structured, criterion-referenced training within platoons. SAFOR ought also to enable "tethered" (Blue SAFOR) exercises in moving, shooting, and communicating by platoon leaders and company commanders.

Whereas USATRADOC once held that all CCTT exercises would employ SAFOR, that command now contemplates using, for certain critical tasks, manned OPFOR vehicular simulators to provide appropriate portrayal of novel threats. Instructors of tactics will no doubt find that their personally controlling SAFOR puts them in a prime position to assure heuristic exercises and cogent After Action Reviews. Tools for increasing an instructor's control over SAFOR, either by selective manning of vehicles, or by software such as the "editor" features of ModSAF, seem inherently advantageous for teaching small unit tactics and techniques. To the degree that embedding ModSAF in Ada impairs that accessibility

and adaptability, to that degree the CCTT SAFOR software's usefulness for training will be degraded.

The panel heard much about extending the automation of SAFOR. Yet, one should not expect Reservists or their trainers often to make extensive use of SAFOR automated to battalion or above. CCTT seems to have been asked to pay too much attention to SAFOR requirements recently generated by Active Component users who seek reliable, highly autonomous performance by SAFOR to support high visibility exercises of brigades and even divisions. The research project (CCSIL) briefed by Commander McBride of ARPA probes more extensive SAFOR autonomy. Yet, extensively aggregated and autonomous SAFOR could slight the behavioral detail at the entity level important for SAFOR utility in the Reserve Components, the very sort of detail controlled by the modular editors of ModSAF. The panel was shown examples of graceful interfaces between SAFOR and large, constructive models of war, interfaces that permitted selective disaggregation of units within the model to the entity level, and interactive resolution of concurrent combat sequences. It seems sensible for PM CCTT to give priority in his SAFOR development to entity level behavior and small unit performance, relying on ARPA or constructive models to furnish higher echelon contexts for CCTT exercises.

The panel did not hear about SAFOR connections between virtual or constructive simulation and subsistent simulation. At the Army's Combat Training Center in Europe, brigades now train with one subordinate unit in the subsistent or real mode of simulation at Hohenfels, while other subordinates participate via a constructive model or via SIMNET. It seems important for the participants afield to be able to observe friendly units operating (in constructive or virtual simulations) within line of sight or within sensor range on either flank, and to conduct reconnaissance of OPFOR threats along avenues of approach that traverse their unit's assigned boundaries. SAFOR could inject entity representations into sensors (e.g., radar or IR), thereby adding to the scope and fidelity of simulation at the CTC, and provisions should be made for such interfaces by PM CCTT.

All the foregoing suggests that CCTT SAFOR, far from being a mature product ready for "type classification," production and fielding, is and should remain for the foreseeable future an evolving assembly of software tools that ought to be highly modular, very accessible, and easily changeable. I recommend that PM CATT reconsider using Ada, but press ahead with the ModSAF re-engineering presently under way.

COMMENTS BY JOHN LAIRD

1. Can the same SAFOR products support both the R&D community and the user community? If yes, what are the products?

Yes. It is critical to both of these communities for them to share products.

R&D products:

Over the next four years, ARPA will be spending millions of dollars on advances in SAF technology including work on intelligent forces; communication, command, and control; wide-area simulation. It is important for the user community to have quick access to this research.

User community (CATTs for example) products:

The development of CCTT SAFOR will lead to the creation of the following:

- A well-engineered SAFOR. This will include the simulation engine as well as the control logic for the SAF entities. Both of these will be valuable to the research community. The first as a platform for interfacing with the DIS environment. The second as a base-line for behavior of platoon and company-level behavior. Other components, such as the terrain database and the inter-visibility calculation software will also provide useful base-lines for research in these areas.
- A database of CISs and their associated audit trails. This information will greatly help anyone trying to do research in automated forces.

The critical question is whether the sharing should occur at the level of individual components, or of complete software systems. I believe that maximal sharing is important (of as complete of software systems as possible). Thus, we want the research community to directly use the products of the user community. I do not expect the user community to be able to directly use the products of the research community, although it is important that pieces of products from the research community should be demonstratable within the user community software.

Rationale for maximal sharing:

- If the research community is not using the products of the user community, the research community may address problems that are tangential to the concerns of the user community.
- The amount of funding available for continued development of SAF environments will be limited. It will be difficult for the research community to have access to a well-engineered SAF system unless it comes from the user community.
- The comparison of research results to a meaningful baseline will be difficult without the research community using the user community SAFOR.
- There is the potential for results of the research community to be demonstrated within the context of the user community SAFOR. For example, if an alternative behavior control system is developed in the research community, it could be tested within the user community (training centers) on real data. In general, sharing a common environment will increase the research communities access to realistic data. If the development goes on in separate environments, there is too great a potential for minor incompatibilities to make integration very difficult.
- The shared expertise in a single software environment will greatly increase the possibility for people to move between the two communities—greatly aiding technology transfer.
- Finally, the SAFOR technology is immature. After the CCTT SAFOR is developed, it will have to be extended for the other CATT programs, as well as other services. Already, many of the requirements for CCTT push it into uncharted waters, making a continual infusion of results from the research community a critical component of its development.

As part of developing a single software environment that is shared between the user and the research community, that environment must have well documented components and interfaces so that the research community can easily modify the software. The current CCTT SAFOR development methodology is consistent with this need.

2. What software approach should be used for CCTT SAFOR considering the needs of both the R&D community and the user communities?

The software methodology, independent of the language, will determine the ability of CCTT SAFOR to meet the needs of the user community (life cycle maintainability, reliability). The selection of the language is clearly secondary. My impression is that the current software methodology is excellent and will lead to good maintainability and

reliability. Thus, we are able to look at other issues, the most important being usability and flexibility by the research community. Here, Ada is inferior to C:

- There is an established base of C programmers in the research community. Most of the major engineering universities teach C in introductory programming courses. C has become the standard for engineering software—all large engineering software packages are being written in C (such as CAD/CAM systems). There is little if any research work done in Ada.
- The entry cost of using a C-based system is much lower than Ada. There are even high-quality free C compilers available on most UNIX systems (GNU).
- The software development cycle of recompiling is much less in C. This is critical in a research environment where software is developed incrementally. Thus, I recommend adopting C for the development of the CCTT SAFOR.

What has to be considered is the life-cycle of the SAFOR project past the delivery of the CCTT SAFOR. Thus, the system must be designed for the infusion of results from the research.

3. What products are available from other programs?

There do not appear to be any other products that the CCTT SAFOR team has missed.

4. What strategy of specific steps should the Program Manager, CATT, take to build community consensus and to produce a product that will support CATT programs other than CCTT?

Nothing special recommended here.

5. What strategy should the Program Manager, CATT, take to implement the CCTT SAFOR?

- Stay the course on the methodology for software development, VVA, building the CIS database.
- Adopt C as the language for implementing CCTT SAFOR.
- Continue to have close ties with the research community such as the relationship between SAIC and LADS.
- Expand the ties to include the work on behavior representation.

6. Other issues:

ModSAF development must continue to be funded over the next 3-4 years to support the research community until CCTT SAFOR is available.

COMMENTS BY DUNCAN MILLER

As the complexity of the SAF development issues became apparent during our discussions, the peer review panel paused to confront the issue of how broadly we should interpret the questions we had been asked to address. We decide to focus on how to maximize the chances of succeeding with SAF development in the long run, rather than on how to fulfill the immediate needs of the CCTT program.

This is a crucial point, because SAF is by no means a mature technology. SAF technology is going to continue to evolve for at least the next several years, and the R&D community will need to be involved in this process for it to be successful. ARPA is already counting on some substantial extensions of SAF technology, especially in the area of representations of decision-making and command and control functions, for WISSARD IFOR and STOW, among other programs. War Breaker will almost certainly make use of these developments, as well.

As SAF technology grows, it will become increasingly important that researchers build their extensions on a solid, accepted base of code. We have already reached the point where it is prohibitively expensive for researchers to reinvent the large body of existing work, even for a program the size of CCTT. Increasingly, they must build on what is there, adding and modifying modules where necessary, but not recreating these modules where changes in functionality are not needed.

In general, the panel was favorably impressed with the ModSAF architecture, with the degree to which the Integrated Development Team had accepted and adopted this architecture, and with the improvements they proposed to add for CCTT. If the approach we saw presented is executed well, CCTT SAF should become the new foundation for SAF development for the next few years. For this to occur, however, the code must be easily transportable, accessible, and affordable to the R&D community, the Combat Developments community, and the Test and Evaluation community. All of these communities share the need to access and modify certain modules within the code for various purposes.

This brings us to the central question of the language in which CCTT SAF is to be written and the tools and support modules it requires. If CCTT SAF is implemented only

in an Ada environment, the number of organizations that can deal with, and modify, the base of code will be very limited. ARPA probably will not be able to use an Ada-based CCTT SAF in its programs, since very few of its contractors are equipped to deal with Ada. This is true from the standpoint of availability of experienced personnel and tools, as well as from a cost and schedule standpoint.

I believe, and it appeared that all of the other panelists also believe, that it would be best if the CCTT SAF were developed and maintained in C, with appropriately rigorous levels of documentation and validation of the algorithms and the code. As new capabilities emerge from R&D programs, they will have to be adapted, tested, documented, and validated for incorporation into CCTT and subsequent CATT programs, but this is a process that must be carried out in any case for a production system.

The other alternative we discussed was maintaining two parallel systems, an R&D version in C and a production version in Ada, trying to keep them as synchronized as possible. Such an approach would prove cumbersome and expensive, at best. At any given point in time, there are bound to be discrepancies in the capabilities of the two systems and in the way particular algorithms are implemented. These discrepancies would make code verifications and performance comparisons difficult, and would make it very hard to interpret the results of tests conducted with differing versions of the software.

If, after considering our recommendations, it is concluded that CCTT SAF development must be done in Ada, we suggest that STRICOM be prepared to undertake at least one additional round of SAF enhancement before CCTT is fielded. Over the next two to three years, we believe that ARPA programs will yield enough important new SAF capabilities that such enhancements will be well warranted.

Finally, it was noted that the development of the code to support some 1200 SAF entity behaviors, which require an average of 10 pages of English narrative to describe, is going to make CCTT SAF the largest representation of intelligent human behavior ever undertaken. This is a daunting prospect. The sheer magnitude of the effort reinforces our point that no other programs in the foreseeable future are going to be able to afford to recreate this corpus of knowledge. They will have to build upon what the CCTT program has achieved. It is imperative that this work be shareable across all of the communities that need access to DIS technology. We think this access will be possible only if the implementation is done in C, the closest thing we now have to a universal language in computer science, but with the Systems Engineering discipline and documentation embodied in MIL STD 2167A.

The above comments focused primarily on the Ada language, programming environment, and tools as barriers that would inhibit or preclude the involvement of large segments of the R&D community in further development of SAF technology. While these are the primary barriers, they are not the only barriers. The injudicious selection of rule-based software could have a similar effect.

I urge that to the extent that rule-generation tools and a rule-processing engine become a key component of CCTT SAF, the selection of these tools should also be considered from the standpoint of other programs and other developers. For all the reasons cited above, these tools will become an integral part of the SAF environment on which others must build. Great care should be taken to ensure that they are available at reasonable cost for a variety of computing platforms. The government should consider acquiring the rights to use these tools in future extensions of SAF, both for R&D purposes and for production use, so that they can be furnished to future developers as part of the SAF software package.

COMMENTS BY ROBERT RICHBOURG

The basic framework I have used to try to answer the questions is that whatever decisions are reached, they must have a favorable impact on the success of the program, both in the long and short terms. Long-term success is the more important of the two and can be measured by the characteristics of the eventual SAFOR product. It must be acceptable by the user community (in the widest sense), be applicable to other programs within the CATT family as well as contribute to basic research questions in the larger community, and be easily extensible as new results become available. While the long-term success is of paramount importance, the short-term results are also important; a short-term disappointment will greatly diminish the chances of a long-term success. The short-term indicators include a program that can be delivered on time, within budget, and, most importantly, can meet the stated needs of the program. Trying to put all these concerns into a few words, the program must be such that the widest community (both R&D and development) develops and feels a sense of ownership for a product that has been engineered using the best available notions from software engineering.

1. Can the same SAFOR products support both the R&D and user communities?

Without question, the deliverables from this program can be used by both communities, and they have wider applicability to commercial enterprise as well. The most important product to come from the CCTT SAFOR effort may well be the documented, electronic codification of behaviors at the various operational echelons (from individual platform through brigade). A credible functional description for military behaviors has been a stumbling block for many programs in the past and could still be one well into the future. The CIS specification should alleviate this problem. Moreover, the CIS will constitute an important source of behavioral description (under a wide set of conditions) that should be of great value in further explorations into understanding general human behaviors. Similarly, the architecture/methodology to utilize the behavioral specifications (the CIS) will have wide interest. While such an architecture will not constitute a proof of the human reasoning process, it will constitute an existence proof of a method that does work (mixing low-level FSA mechanisms with high-level, rule-based techniques). Again,

the architecture as well as the CIS themselves should prove useful in any area that requires a model of human behavior.

The actual software (source and executables) may have less widespread applicability than the CIS and CIS architecture. However, they should still prove useful to the R&D community as a testbed for the exercise of new ideas. Applicability to the military user communities will be more direct. Military education could use SAFOR as the players who reenact past battles, as we have already seen in 73 Easting. The educational benefit could be enhanced by a SAFOR enabled "what-if" capability for history. SAFOR can be used for other educational purposes, such as trying to teach military students how to identify an OPFOR (the SAFOR) center of gravity. Clearly, the training community can benefit from continued use of the SAFOR. New doctrine and systems can be tested and refined based on SAFOR (both as OPFOR and friendly forces). The analysis community already relies on a form of (aggregated) SAFOR that will be improved by producing more capable computer-controlled force. The logistics community should rely on SAFOR to a large extent in the exercise of both support and mobility planning. As an example, exercise of a port clearing plan has little training value for human equipment operators but has high value for those who must devise and refine the plan. SAFOR could be used for such purposes. SAFOR can also prove helpful in the examination of operational and strategic questions, given a sufficiently high level CIS specification. As an example, suppose SAFOR are used to model an ally (on the friendly flank) who uses tactics typically associated with the OPFOR (the Syrians in Desert Storm). SAFOR could be used to answer many questions in such cases, including those that revolve around an ally who suddenly decides to change allegiance mid-battle.

Given a modification in some of the CIS specifications, the SAFOR could also apply to commercial purposes. Airborne SAFOR could be used to train air traffic controllers. DI SAFOR could help train law enforcement personnel for riot, hostage, and other situations (Waco, Texas?). Vehicular SAFOR would be useful to city planners when designing transportation networks.

In short, the CCTT SAFOR effort should find application in many areas.

2. What software approach should be used for CCTT SAFOR, considering the needs of both the R&D and user communities?

All communities will best be served by a software system that strictly adheres to the commonly accepted principles of "good" software engineering practice. There are several

programming styles that could be applied to this program including procedure-oriented, object-oriented, logic-oriented, rule-oriented, or constraint-oriented. The developers in the program are charged with producing a large (over 100K SLOC), "industrial-strength" program that will function in a complex domain. Without producing a text on software engineering, I simply state the commonly accepted view that the object-oriented style is best suited for such programs. Given that the object-oriented style will be used, there are several characteristics that should be required in the implementation language. These include provisions for abstraction, encapsulation, modularity, hierarchical capabilities, strong typing, concurrency support, and enabling persistence. While the implementation language need not support all of these, supporting most of them will be helpful in using the object-oriented style.

Other constraints on the implementation language are more pragmatic. It must generate fast, efficient code. The goals for the number of entities to be "on the net" are very ambitious and cannot be met by an implementation language that makes too many sacrifices in execution speed. The language must have a wide base of support. The R&D community is concerned about the issues of additional cost, both in acquisition of new software and in training people to use new software. In summary, the perfect language for the CCTT SAFOR program would support software engineering principles (by reliance on the object-oriented paradigm), produce fast, efficient code, and be widely available in both the production and R&D communities. Both the R&D and the production communities must have access to the language. The product of the production effort must be accessible by the R&D community so that they can have a testbed to generate new results (which will be transferred back to the production environment). Actual software transfers from the production community to the R&D community while concepts and methodology transfer from the R&D community back into the production environment.

From the presentations at the peer review, it seems that the R&D community has a preference for C, a language that is not object-oriented, based primarily on the wide availability of both C environments and programmers. The IDT advocates use of Ada, an object-based but not object-oriented language, based on concern for software engineering issues. Remember that the dominating factor in the quality of code production is the skill/talent of the individual programmer. A talented software engineer will produced well-engineered code given any high-level language. The real software engineering issue here is deciding to what degree does the chosen language encourage the average programmer to produce a well-engineered product. While use of Ada will not, by itself, guarantee a good

software engineering effort, it was designed to encourage and enforce the use of good software engineering practice as it was understood in the late 1970's and early 1980's. The same can not be said of C; use of good software engineering principles in this language is more a matter of management enforcement.

In an effort to recommend an implementation language, I have considered four alternatives; Ada, Ada 9X, C, and C++. Given any language choice, I assume that the IDT will continue to rely on the same set of tools to perform OOA and OOD, so that the degree to which any language "dovetails" with the OOA/OOD result is also an issue.

a. Ada:

Ada is object-based but not object-oriented (as it does not support inheritance) and it is not well established within the R&D community. Public domain Ada compilers are available (Gnu project) but these are not subject to the formal validation process. Thus, compiler acquisition costs are not large issues. However, training costs to produce Ada programmers in the R&D community will still have an impact (more time and effort than dollars). Ada code is generally slower than equivalent C code. An Ada implementation would fit well with the products from IDT OOA and OOD. Ada was designed to encourage programmer use of software engineering principles.

b. Ada 9X:

Ada 9X is the fully object-oriented improvement of Ada and should thus be more useful in the object-oriented style of program development. Public domain Ada 9X compilers should be available by 31 December 1993 (again from the Gnu program), but these will not be subject to formal validation. Programmer familiarity within the R&D community remains an issue. Further, Ada 9X is a new language that does not enjoy a wide support base in any community and has not yet been subject to public scrutiny to ensure an error-free compiler or environment. I assume that Ada 9X execution speed is similar to that of Ada code. An Ada 9X implementation should fit exactly with the results of IDT OOA and OOD. Ada 9X should encourage a high degree of good software engineering practice.

c. C:

C is procedure-oriented, not object-oriented for many reasons. C is widely supported and available in both R&D and commercial communities. C code generally executes very fast. As C is not object-oriented, it may not align well with the results of

IDT OOA and OOD. C was not designed with promotion of good software engineering principles in mind and, in many cases, the "shortcuts" available in C can actually encourage the programmer to eschew established software engineering practice, frustrate management attempts to enforce software engineering principles, and produce runtime errors that are very difficult to locate.

d. C++:

C++ is object-oriented and enjoys wide use and support in both the R&D and commercial communities. C++ code generally executes fast, but more slowly than conventional C code. As C++ is fully object-oriented (although it does not support persistence) is should fit well with the results from IDT OOA and OOD. While not designed with software engineering support in mind, C++ is object-oriented, which should aid in quality software production. Also, as C++ should be able to reasonably reflect the results of OOA and OOD, it should encourage good software engineering practice.

Recommendation: Select C++ as the implementation language.

A second issue of concern regarding the implementation involves the use of the COTS product RTWorks to implement the rule-based component of the software. Again, acquisition and training costs for the R&D community cause some problems. An alternative in this area is the C Language Integrated Production System (CLIPS), a GOTS product that was built at NASA to allow the implementation of AI-type systems (RBS and expert systems) on conventional hardware. A companion product is CLIPS Object-Oriented Language (COOL) which allows object-oriented concepts to be used with the CLIPS environment. These products are available free of cost to government agencies and at nominal cost to others (approximately $350). While a firm recommendation to use CLIPS cannot be made without specific knowledge of the IDT expected benefits from using RTWorks, it is an option that deserves some examination.

3. What products are available from other programs that could also be useful in developing CCTT SAFOR?

In general, the DMSO Survey of SAFOR and the IDT together seem to have covered this question well. One small addition to those systems already examined is CLIPS, as discussed above. Also, encoding the CIS is a large project in constructing a knowledge base of rare size (very large), high complexity, and intended for wide and general use. Doug Lenat's Cyc project at MCC may offer some insight into the issues and

methodology involved when constructing such a base of knowledge that should be generally useful.

4. and 5. How should the PM CATT proceed to build community consensus and build a CCTT SAFOR that will be useful well into the future?

In general, the program seems to be progressing exceptionally well and most recommendations in this area involve continued or expanded use of practices that have already proven successful.

a. Continue to use the spiral development technique.

b. Using people rather than documentation to effect information exchange is a success. Continue the relationship between the IDT and LADS and consider opportunities to include personnel from the R&D community (visiting scholars) in the program.

c. The SUMEX worked well and should be continued on a regular basis. Again, consider opportunities to involve selected researchers from the R&D community in the program. This could have wider impact than just for the CCTT SAFOR development.

d. Take a more active role in funding research. At some point, ARPA emphasis will be shifted to other areas and CCTT should have a mechanism in place to fill the void.

e. Insist on tightly specified, well documented interfaces for the IDT software. Similarly, emphasize modularity to encourage "plug-in" use of the CCTT SAFOR products.

f. Insist on strict standards for data abstraction; maintain all data separate from any code that uses it. Apply this to the knowledge base created as the CIS as well. Documentation is necessary.

g. Continue soliciting review and feedback from the R&D community both to enhance their feeling of ownership for the program as well as to get their insights.

h. Insist that architecture and methodology are as well documented as the code itself. These are major reusable products.

i. Place a strong emphasis, including incremental user review, on the construction of system interfaces. A poor interface can ruin an otherwise exceptional product while a strong interface can support a marginal effort. The interface deserves a lot of attention and effort.

j. Maintain the same high levels of effort and productivity and keep up the great work!

APPENDIX B

BIOGRAPHICAL SKETCHES OF PANEL MEMBERS

BIOGRAPHICAL SKETCHES OF PANEL MEMBERS

Dr. Philip S. Anton is a member of the technical staff at the MITRE Artificial Intelligence Technical Center. His research interests include behavioral representation, distributed simulation, computational neuroscience, and neural network modeling. Dr. Anton received his B.S. in Engineering from UCLA, and his M.S. and Ph.D. from the University of California at Irvine in Information and Computer Science, specializing in Artificial Intelligence and Computational Neuroscience. He is a member of IEEE, ACM, INNS, and CPSR.

Peter Brooks is a member of the Strategy, Forces, and Resources Division of the Institute for Defense Analyses in Alexandria, Virginia. He received the A.B. degree in mathematics from Princeton University, and the M.S. and Ph.D. degrees in mathematics from Stanford University. Prior to joining IDA, he held a research position with the Department of Defense. His recent work includes contributions to the development of new analytic methods for use with SIMNET and DIS exercises, a workstation-based smart minefield simulator, and test procedures for Semi-Automated Forces.

Gen. Paul F. Gorman, U.S. Army (Retired), is Visiting Professor of Computer Science, University of Virginia, and a consultant on military training for the Institute for Defense Analyses, and for the Defense Science Board. He has contributed to the development of virtual tactical engagement simulation from its inception.

John E. Laird received his B.S. from the University of Michigan in 1975 and his Ph.D. in Computer Science from Carnegie Mellon University in 1983. He is currently an Associate Professor in the Electrical Engineering and Computer Science Department of the University of Michigan and Director of the Artificial Intelligence Laboratory. His primary research interests are in the nature of the architecture underlying artificial and natural intelligence. His work is centered on the development and use of Soar, a general cognitive architecture. Most recently he is the principal investigator on the Soar/IFOR component of WISSARD/IFOR. The goal of this project is to develop intelligent forces within tactical air-to-air engagements.

Duncan C. Miller holds four degrees from MIT, including the Doctor of Science in Mechanical Engineering. His principal areas of study included control theory, human operator performance modeling, human factors, and perceptual psychology. At Bolt Beranek and Newman Inc. (Cambridge, MA, 1963-1993), Dr. Miller formed and managed the group that developed the protocols and software for SIMNET. He is a member of the Distributed Interactive Simulation Standards Steering Committee, which refined the SIMNET protocols into IEEE Standard 1278-1993 (Standards for the Interoperability of Defense Simulations). He has served on a number of advisory committees, boards, and panels, including the Naval Research Advisory Committee Panel on the Impact of Advancing Technology on Exercise Reconstruction and Data Collection (1990-91), the Defense Science Board Task Force on Simulation, Readiness, and Prototyping (1992), the Congressional Office of Technology Assessment Defense Modeling and Simulation Project Advisory Panel (1993-94), and the Air Force Science Advisory Board Joint Modeling and Simulation Systems Review Panel (1994). Dr. Miller is currently leading a small group of experts at MIT Lincoln Laboratory, facilitating cooperation and technical exchange across government programs in such areas as Distributed Interactive Simulation standards and architectural issues; data flow management in large interactive networks; information management for large, multisite exercises; C^3 simulation protocols; and other issues involved in interfacing virtual, live, and constructive simulations.

Robert Richbourg was commissioned as an officer in the Regular Army in 1976 after graduation as a Distinguished Military Graduate from the ROTC program at Wake Forest University. He has served on battalion staff sections in various positions and commanded a HAWK missile battery in the Federal Republic of Germany. His military schooling includes the Air Defense Artillery Officer's Basic and Advanced Courses as well as the Command and General Staff College. In 1984, he earned a Computer Science Master's Degree as the top graduate in the Computer Science curriculum at the Naval Postgraduate School. He became the first Computer Science Ph.D. graduate of the Naval Postgraduate School in 1987. Lieutenant Colonel Richbourg is currently an Academy Professor at the United States Military Academy, West Point, where he has served as the Director of the Office of Artificial Intelligence Analysis and Evaluation since 1988.

APPENDIX C

SURVEY OF SEMI-AUTOMATED FORCES BRIEFING (MITRE CORPORATION)

Survey of Semi–Automated Forces

Sponsored by
Defense Modeling and Simulation Office

Presented by
Lashon Booker

27 October 1993

Contents

- Purpose
- Basic SAFOR Characteristics
- Findings
- Issues
- Recommended SAFOR Research Areas

Purpose of Survey

- Provide information baseline about systems that purport to meet SAFOR requirements
  - Existing
  - Planned
  - R & D efforts

DMSO long-term goal: Define a common architecture for SAFOR development

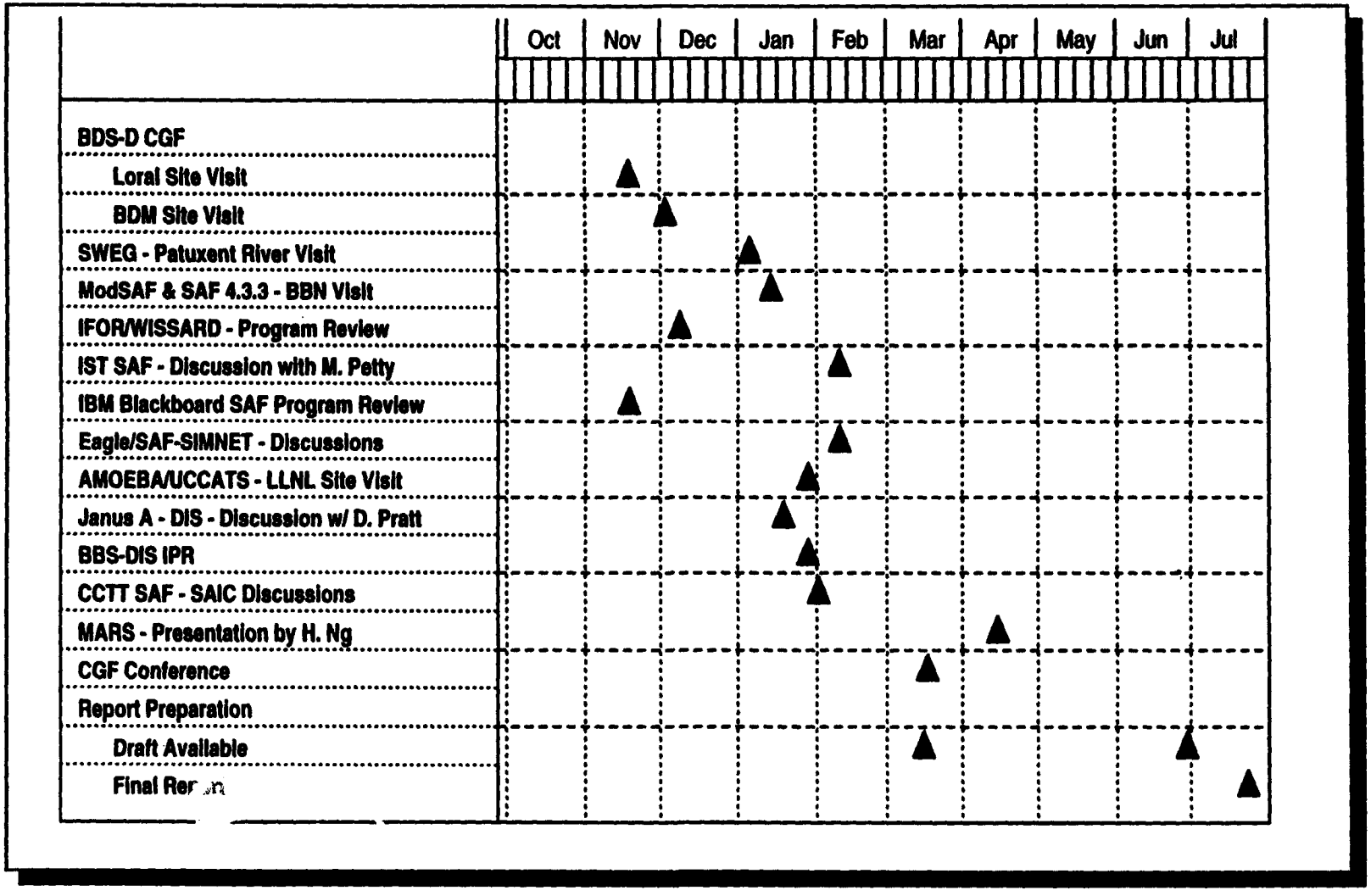
Basic SAFOR Characteristics

- DIS compliance
- Real time interface
- Entity-level combat representation
- Credible surrogate for behavior of manned simulators

Scope of Survey

- BBN SAF 4.3.3
- MODSAF 1.0
- IST SAF
- BDS-D CGF
- SWEG/Suppressor
- CCTT SAF
- WISSARD/IFORS
- Janus A
- AMOEBA/UCCATS
- JCM
- IBM blackboard
- EAGLE/SAF—SIMNET
- RESA—DIS
- BBS—DIS
- MARS

Survey Schedule



Findings: General Impressions

- SAFOR representation of combat is limited:
  - Army
  - Primarily combat and combat support
  - Nascent combat service support representations
  - Navy and Air Force representation limited
  - Mostly low-level (vehicle up to battalion)
  - No explicit models of command and control that extend across all echelons
  - No mobilization

Findings: General Impressions

- Limited threat representation:
  - Classic Soviet tactics
  - Unclassified threat data only
- Development community
  - Supported by small customer base
  - Guided by informal requirements
- No fully object-oriented approach

Findings: Good Ideas

- Explicit representation and capture of command and control decisions
- Use of standard messages and formats for command and control at several echelons
  - Plan and order-based hierarchical command and control
  - Communication via messages
- Configuration-time substitution of different behavioral representations of entities (e.g., hardware missile seeker vs. S/W model of missile seeker)
  - Well-defined run-time interfaces for models
  - Granularity of SAFOR entities to allow substitution at component level

Findings: Good Ideas

- 3-D view controllable from SAFOR human-system interface
- Persistent, distributed storage of state information supporting:
  - Fault tolerance
  - Recovery/replay mechanism
  - Load balancing
- Arbitration schemes for resolving competing goals
  - Currently limited to platform level

Findings: Things We Didn't See

- Capstone architecture
- Broad user community involvement
  - No formal requirements
  - Little transition to analysts and materiel developers
  - No systematic, structured mechanism for involving the end-user
- Balanced representation of joint operations
  - Heavy emphasis on Army ground combat
  - Some portrayal of combat service; nascent portrayal of combat service support
  - Very limited Air Force
  - Little Navy (some NAVAIR, some surface ships)

Findings: Things We Didn't See

- Life cycle support
  - Plans
  - Tools
  - Cost estimates
  - Requirements analysis and tracking
- Object Oriented implementation
- Ada implementation

Recommended SAFOR Research Areas

- Behavioral Representation of Decision Making
  - High-level arbitration of competing goals
  - Under uncertainty
  - Based on anticipation of consequences
  - Adaptive behavior
  - Potential roles for existing cognitive models of command and control

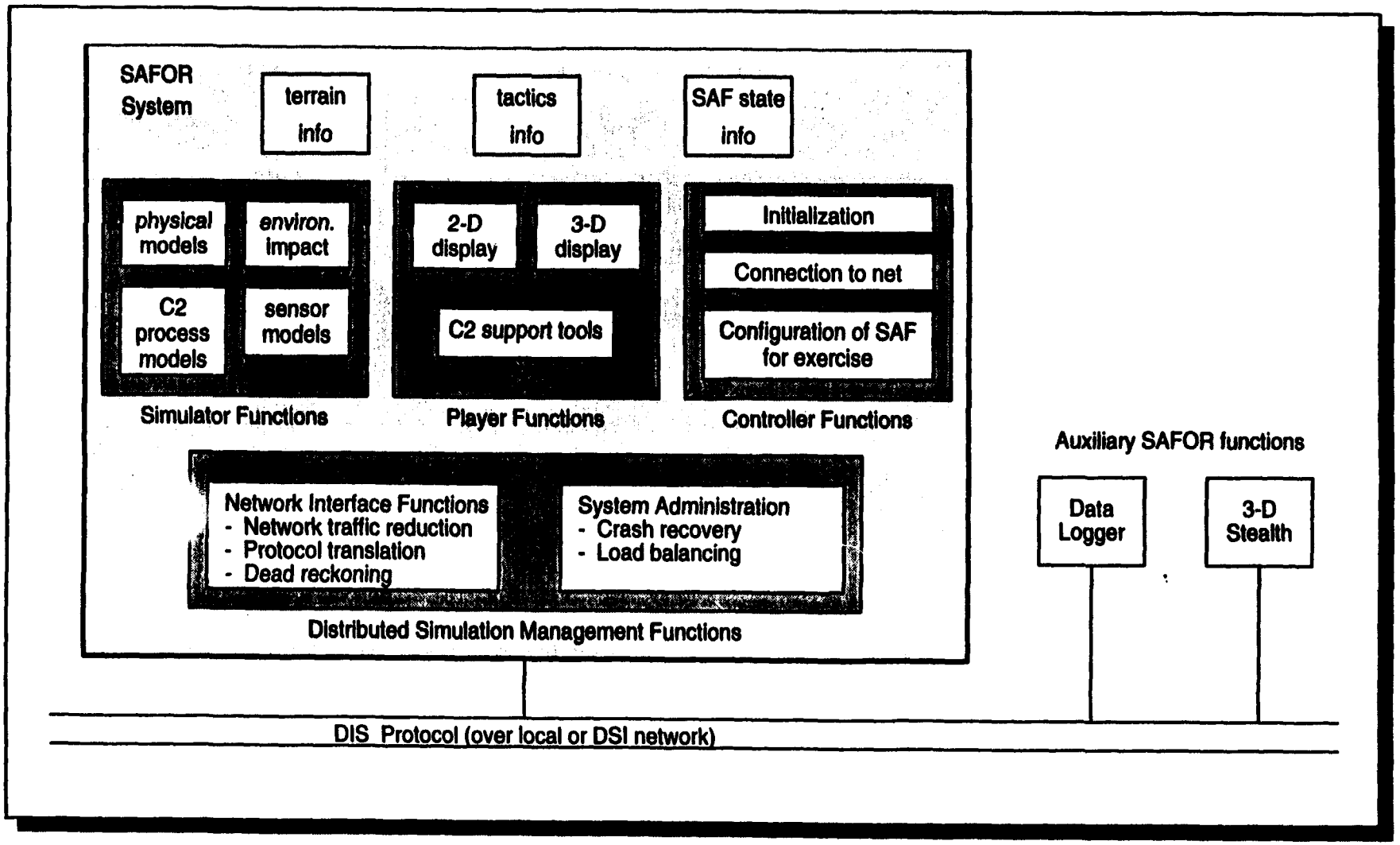
Recommended SAFOR Research Areas

- Exercise Support
  - Scenario generation tools
    - Automatic plan cascading
  - Exercise analysis
    - Tools for querying and filtering logger data
    - Identification of critical events
    - In-progress summarization of battlefield events
    - Synthesis of logger data from distributed sources

Recommended SAFOR Research Areas

- Algorithms for using terrain data
  - Movement
  - Detection
- SAFOR specific PDUs
  - Issue: Should data exchanges between SAFORs be handled with PDUs

Notional View of Objective SAFOR System



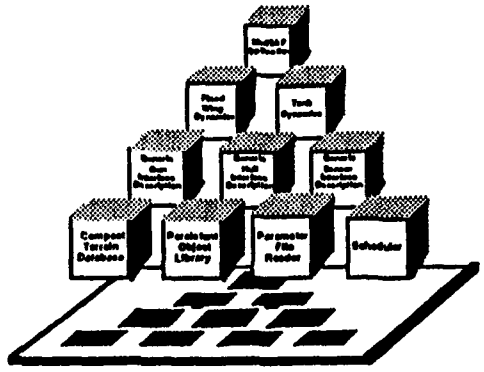
Features of Objective System

- Modular, reconfigurable representation
  - Physical, C2, sensor, and environment impact models
  - Well-defined generic model interfaces
- Flexible, maintainable, open software architecture
  - Model constructs explicity address issues of language design
  - Support for formal tracing of requirements, algorithms, and data

APPENDIX D

ModSAF: AN OVERVIEW BRIEFING (LORAL CORPORATION)

Project ModSAF



A Modular Solution for Semi-Automated Forces

ModSAF
An Overview

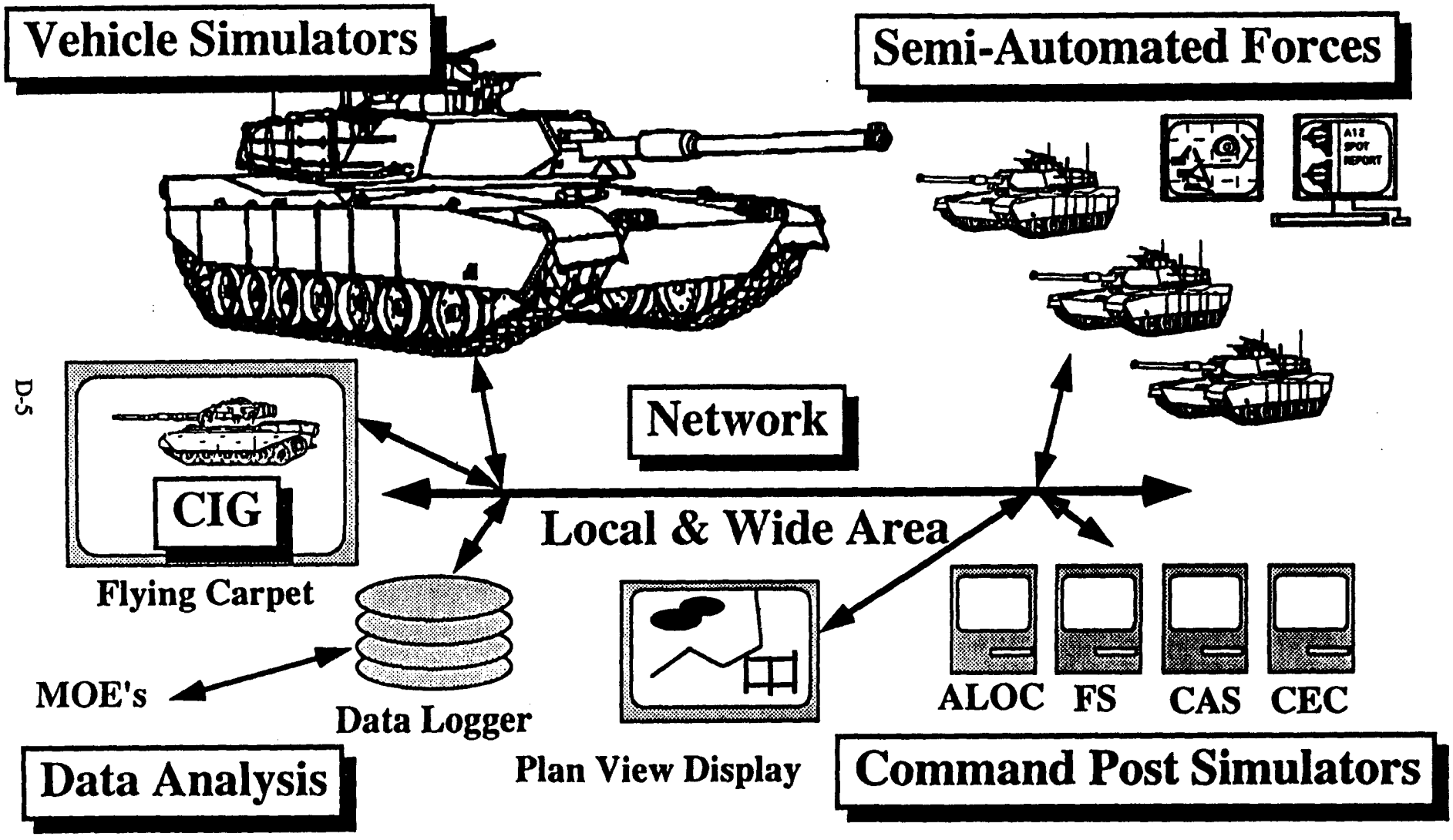
A Generalization and Re-design of SAFOR
that Supports all the Capabilities of
the SIMNET and ODIN SAFOR Systems
and Provides a Solid Foundation for Future Extensions

LORAL
Advanced Distributed Simulation

Outline

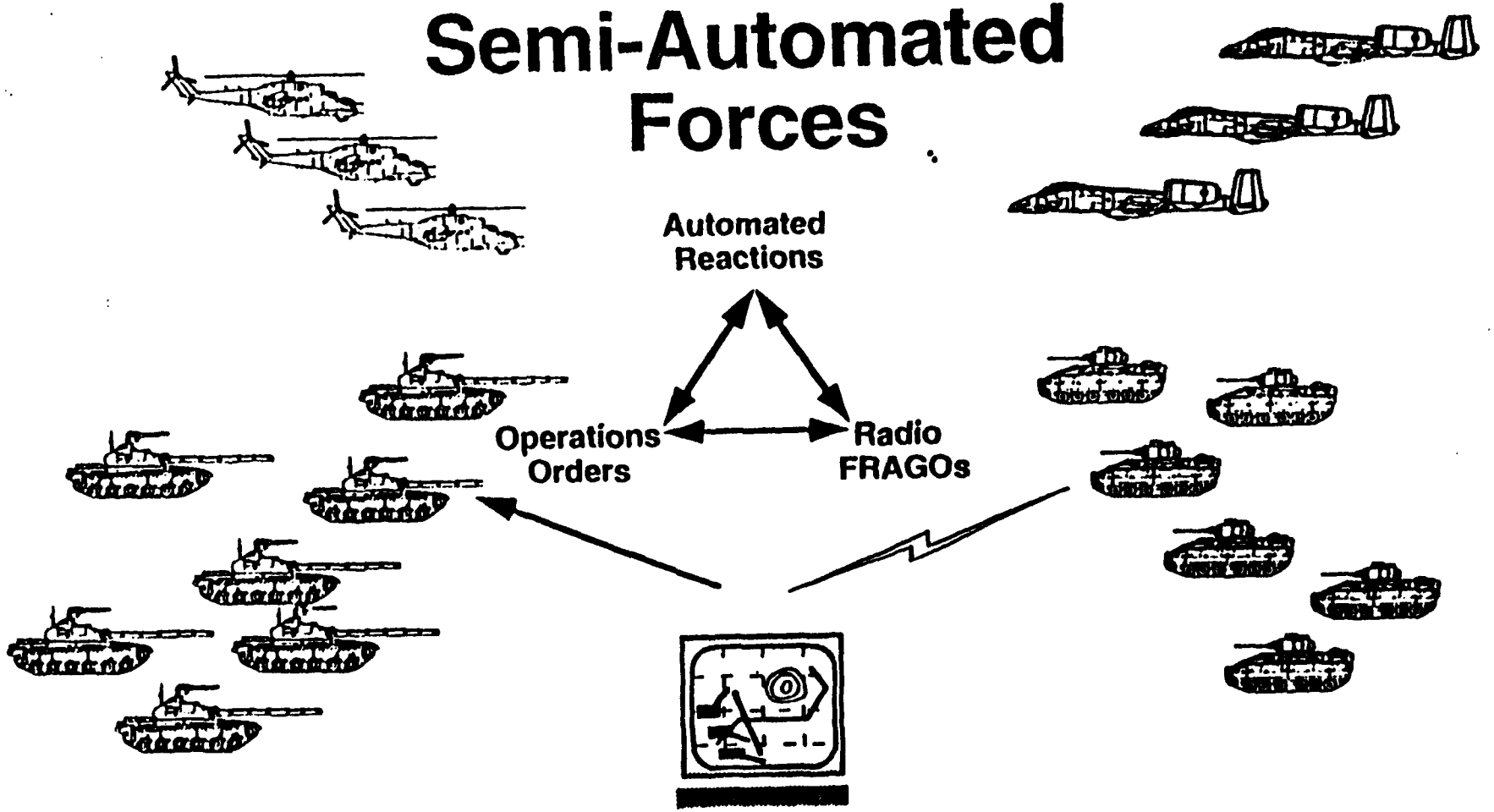
- ModSAF Overview
- System Architecture
- Persistent Object Database
- Behavioral Representation
- Physical Models
- Use of Terrain
- Interfacing to ModSAF
- Versions

Distributed Interactive Simulation



LORAL

Semi-Automated Forces



SIMNET SAF	AGPT SAF	Project ODIN	73 Easting Re-enactment	ModSAF 1.0
1986	1989	1990	1991	1993

LORAL

SAF Applications

- **Training Troops and Commanders:** 2 Bn, 6 Co, 2 mobile, 1 aviation SIMNET and 16 AGPT sites
- **Tactics and Material Development:** 2 sites, CVCC
- **Test and Evaluation:** FAADS
- **Electronic Sandtable:** ODIN
- **Historical Reenactment:** 73 Easting
- **Building block for Intelligent Forces:** WISSARD
- **Link between Constructive and Virtual Simulations:** BBS, AWSIM

LORAL
Advanced Distributed Simulation

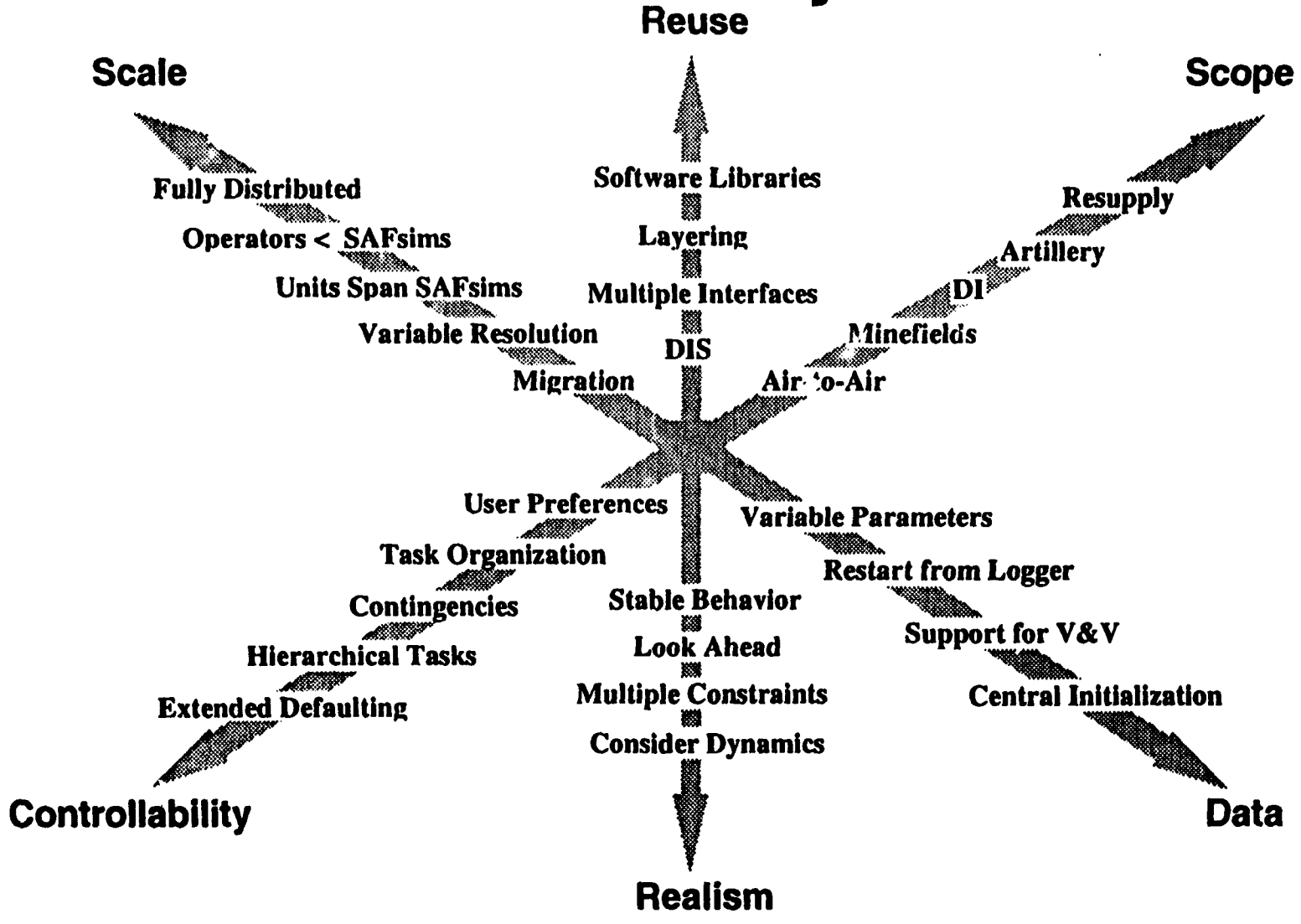
Sample SAF Entities

	U.S. Entities		Other Entities	
Tanks	M1	M1A1	T72	T62
Mechanized	M2	M3	BMP1	BMP2
ADA	FAADS		SA-9 SA-7BMP	ZU-23/4
FWA	A10		Su-25 Frogfoot	
RWA	AH-64	OH-58D	Mi-24 Hind	Mi-28 Havoc
Command	M113 HMMWV	M577	ACRV	MTLB
Supply	M977	M978	URAL 375F	GAZ66
Maintenance	M88			
Artillery	M109	M106	2S1	D20
Troops	U.S. Infantry		Iraqi Infantry	
Missiles	Stinger	Patriot	Scud/MAZ-543	

ModSAF Objectives

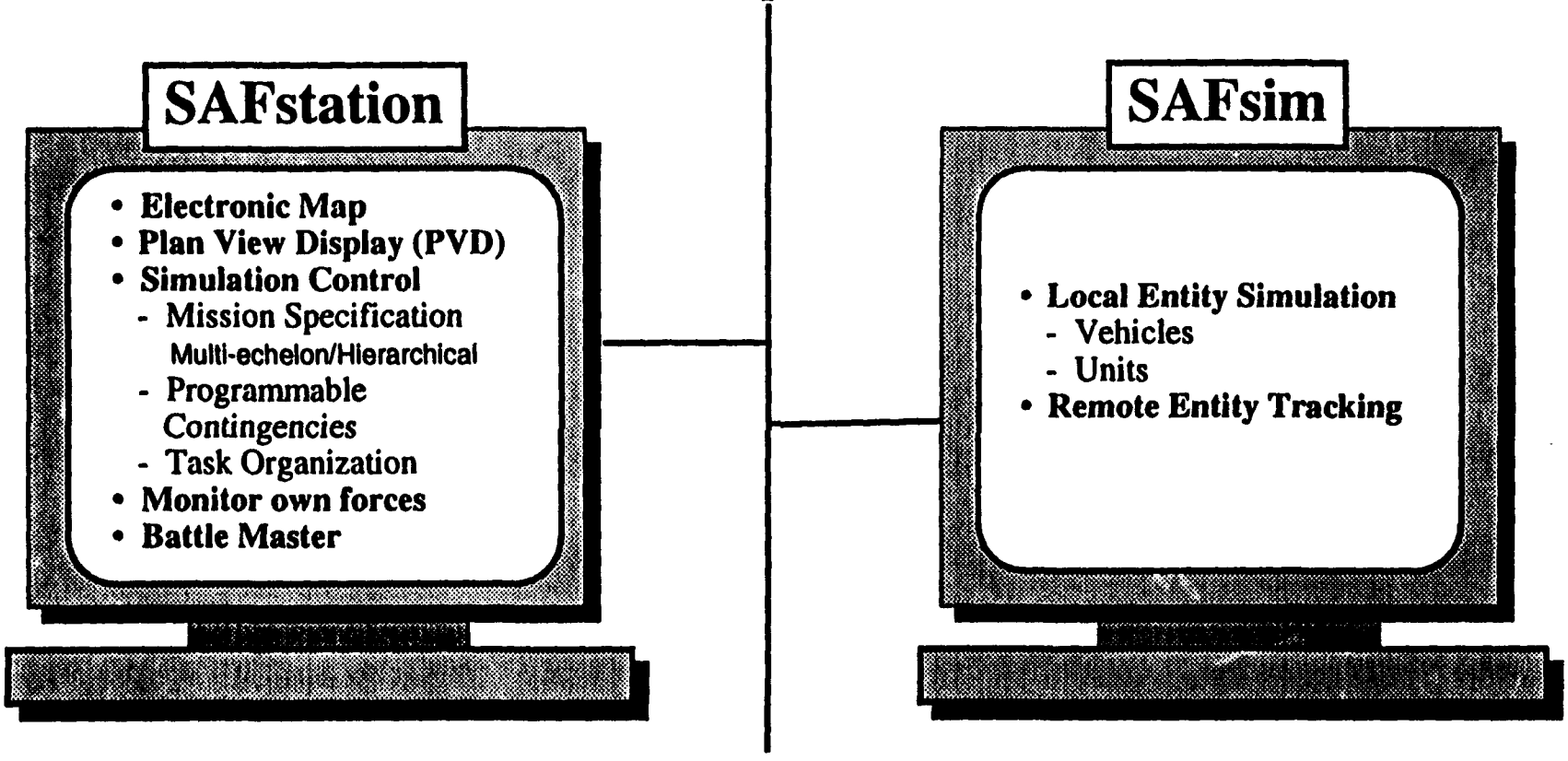
√ **Capacity**	More vehicles, larger units
√ **Resolution**	More accurate models, AMSAA and TRAC
√ **Scope**	Add BVR ATA combat, CSS, FS, DI
√ **Controlability**	Reduced operator workload, more flexibility, higher echelon control, more sophisticated missions
√ **Reactions**	More sophisticated and controllable. Generalized control mechanisms make it easier to add new reactions
√ **Observability**	Recordable, inspectable, decision logic
√ **Setup**	Centralized with ability to restart without checkpointing
√ **Extensibility**	Modular architecture, easier to add/replace component models, platforms, and weapons
√ **Interface**	A block for other DIS simulations: generic physical, task, and mission interfaces
√ **Protocols**	DIS, SIMNET
√ **Hardware**	SGI, MIPS, Sun

ModSAF 1.0 Objectives



LORAL
Advanced Distributed Simulation

SAF Components

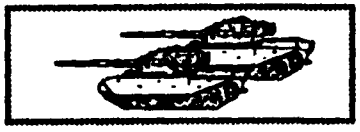


- **Opposing Forces**
- **Flanking and Supporting Forces**
- **Subordinate Vehicles**—*commanded by manned individual simulator*

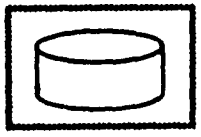
LORAL
Advanced Distributed Simulation

ModSAF Network Architecture

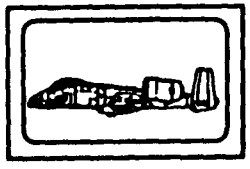
SAFstation
allows users to monitor and control SAF entities



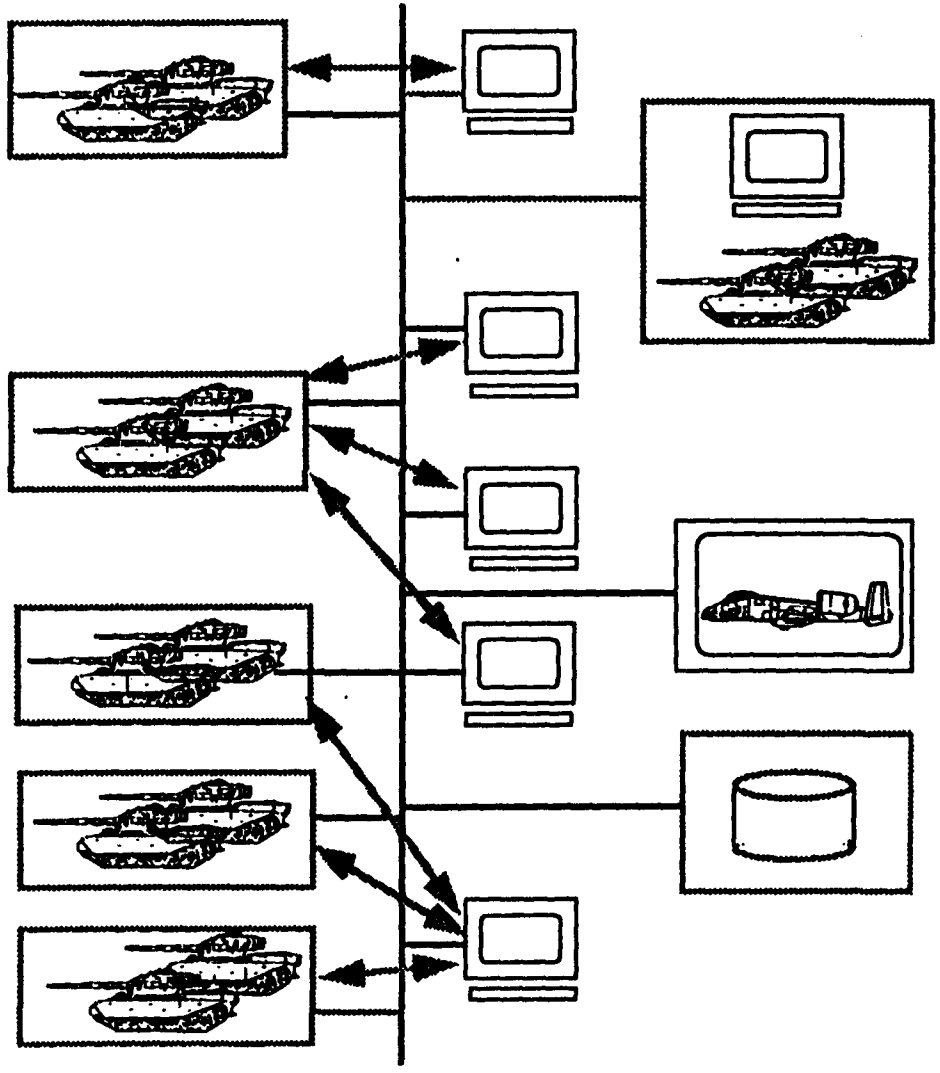
SAF sim
performs simulation of units and platforms



SAF-logger
records/replays network traffic



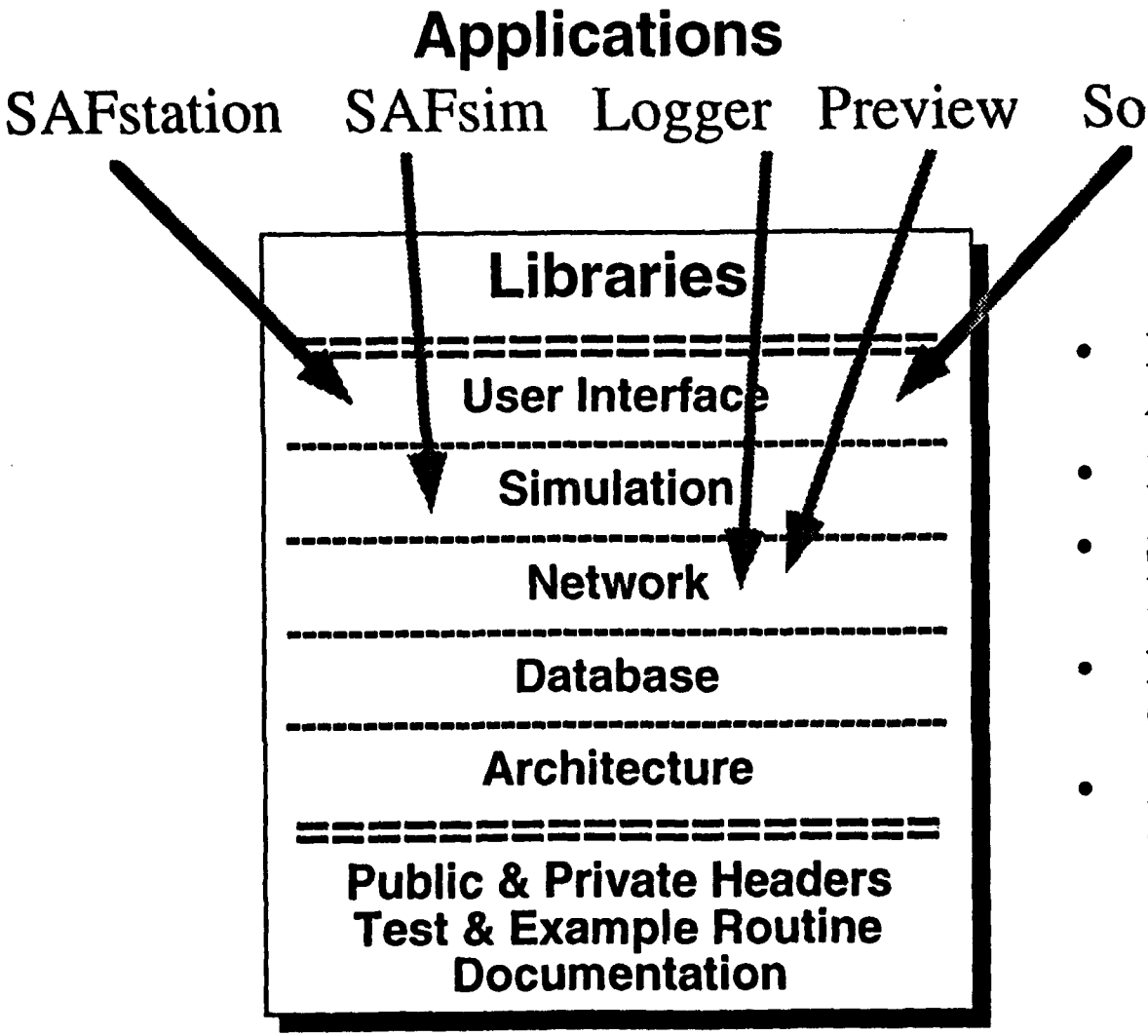
Preview
provides perspective view



Network

LORAL
Advanced Distributed Simulation

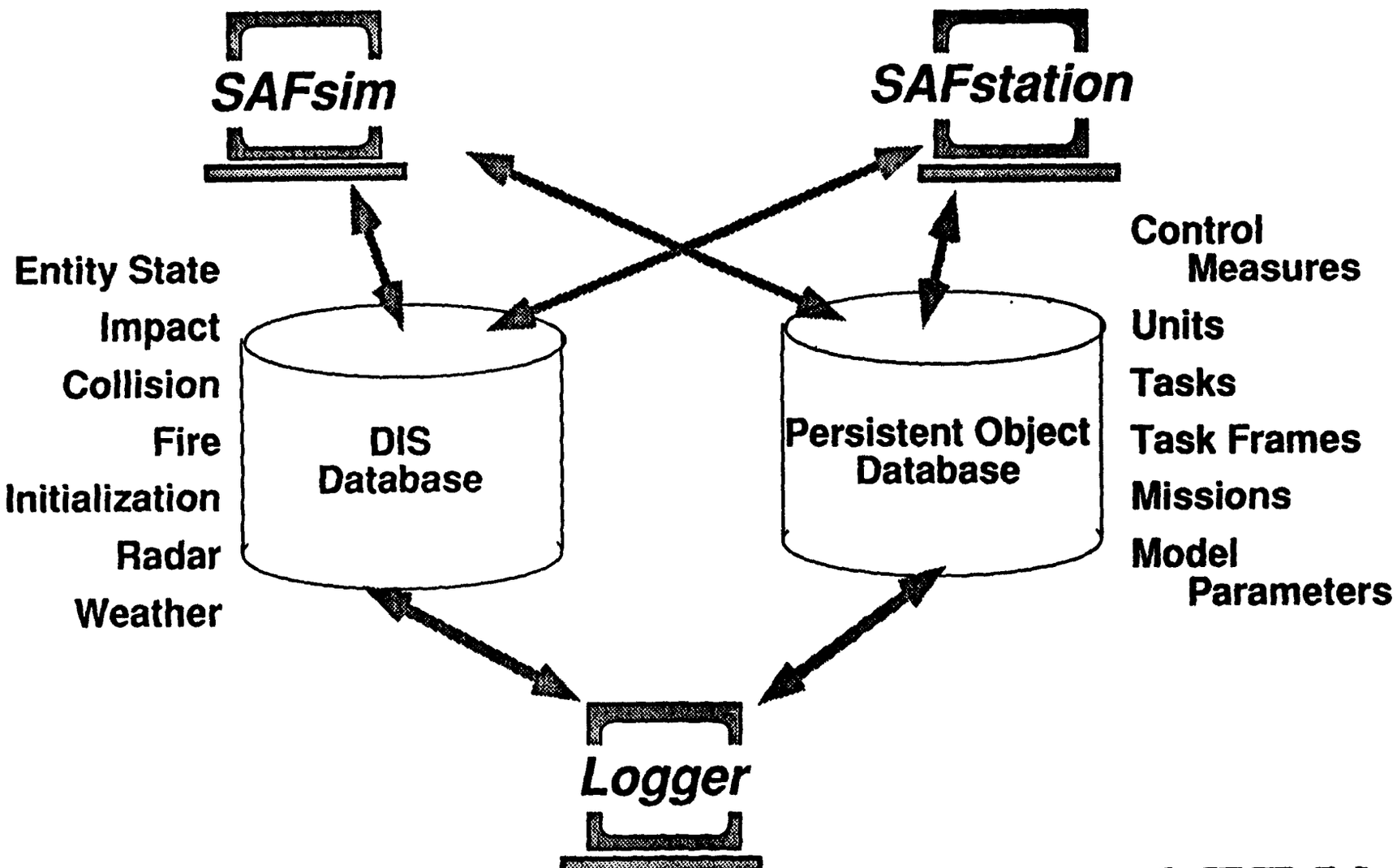
ModSAF Software Architecture



- Library Software Archives
- Layering
- Skeleton "Main" Program
- Data-Driven Object-Based Simulation
- Automated Dependency Grapher

LORAL
Advanced Distributed Simulation

Shared Databases

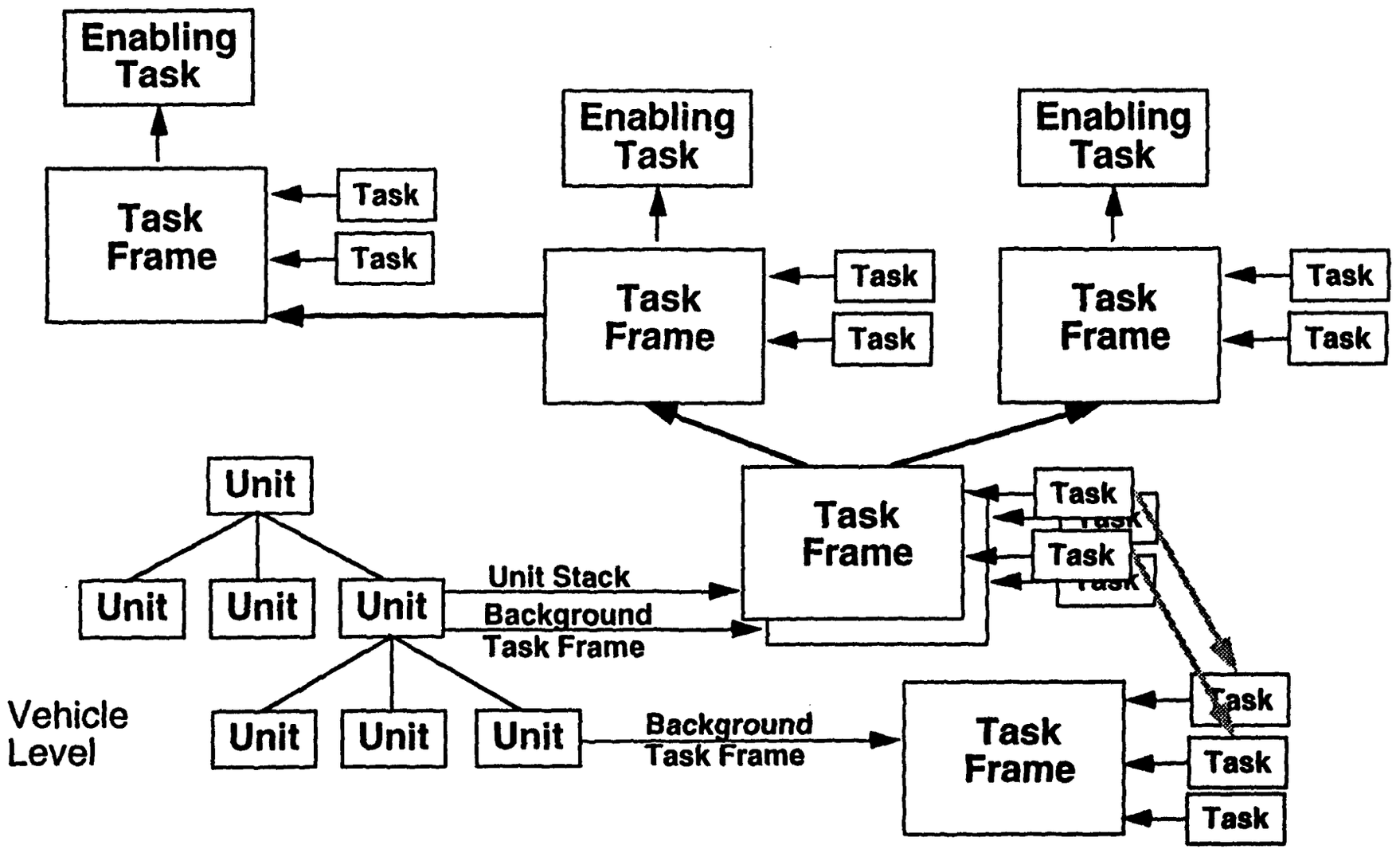


LORAL
Advanced Distributed Simulation

Persistent Objects

Control Measure	Point, line, area, or route
Unit	Entity or unit
Task	Individual behavior—examples include: moving object collision avoidance, platoon bounding overwatch
Task Frames	Collections of tasks which execute in parallel and form a component of a mission
Task Frame Stack	Collection of task frames which a unit is currently executing; only one task frame is active
Missions	Are represented by a network of task frames
Overlays	Organize persistent objects representing—orders of battle, intelligence information, missions
H-Hour	Used to synchronize mission execution by different units

Persistent Object Relationships

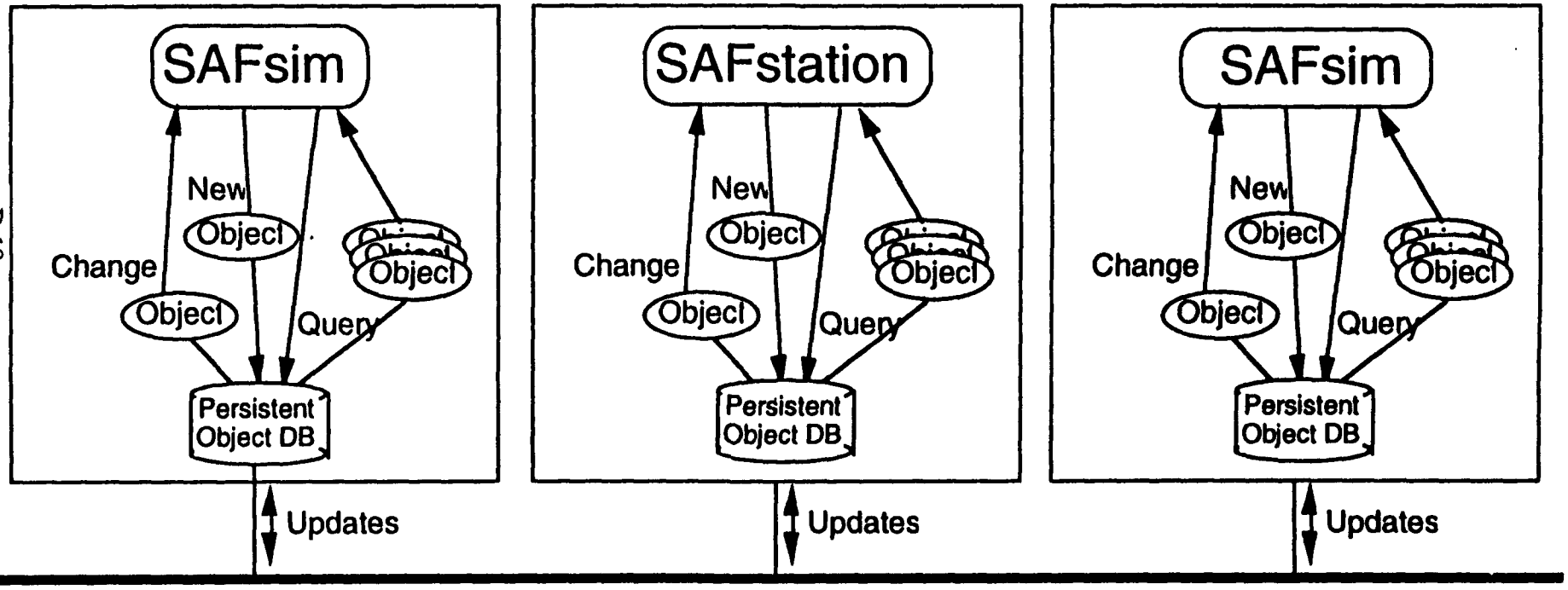


LORAL
Advanced Distributed Simulation

PO DB Usage

- SAFstations and SAFsims Modify the DB Contents as Required, but do not React to the Modification Until the Change is Returned via an Event
- Remote Changes Appear Identical to Local Changes
- Application Software does not need to Distinguish Between Local and Remote Changes
- All C2 Information Communicated through the DB
- Allows Flexible Process Configurations (e.g., SAFstation and SAFsim in One Process, or in Different Processes Separated by a Network)

Persistent Object Database Operation



LORAL
Advanced Distributed Simulation

PO Database Properties

- Every SAFsim or SAFstation has Access to Local Copy of Entire DB
- Every SAFsim or SAFstation can Create, Change or Delete Objects, and can Query and Search the DB
- Every SAFsim or SAFstation is Notified via Events When Objects are Created, Changed or Deleted
- SAFsim's and SAFstation's are Objects in the DB. These Objects have Properties (load, GUI, Sim, etc.)
- DB is Self-correcting Despite any Network Unreliability (This is Analogous to DIS Protocol)

Persistent Object Protocol

- **Broadcast Protocol**
  - 30 sec Rebroadcast Supports many Objects
  - Source-Based Filtering Reduces Steady State Packet Rate
- **PDUs**
  - Simulator Present
  - Object Present
  - Delete Objects
  - Nomination
  - Describe Object
  - Object Request
  - Set World State
- **Properties**
  - Load Balancing during Vehicle Creation
  - Migration of Vehicles between Hosts
  - Fault Tolerance via Migration when Simulator Present Times Out

Persistent Object Database Implications

- Allows Command and Control of SAF to be Recorded with Physical Events for Later Analysis and Validation
- Logged Data Provides Enough Information to Restart SAF
- Operator can Command Larger Units Distributed Across SAFsims
- Units can be Transferred Between Workstations
- Fault Tolerance and Vehicle Migration Between SAFsims

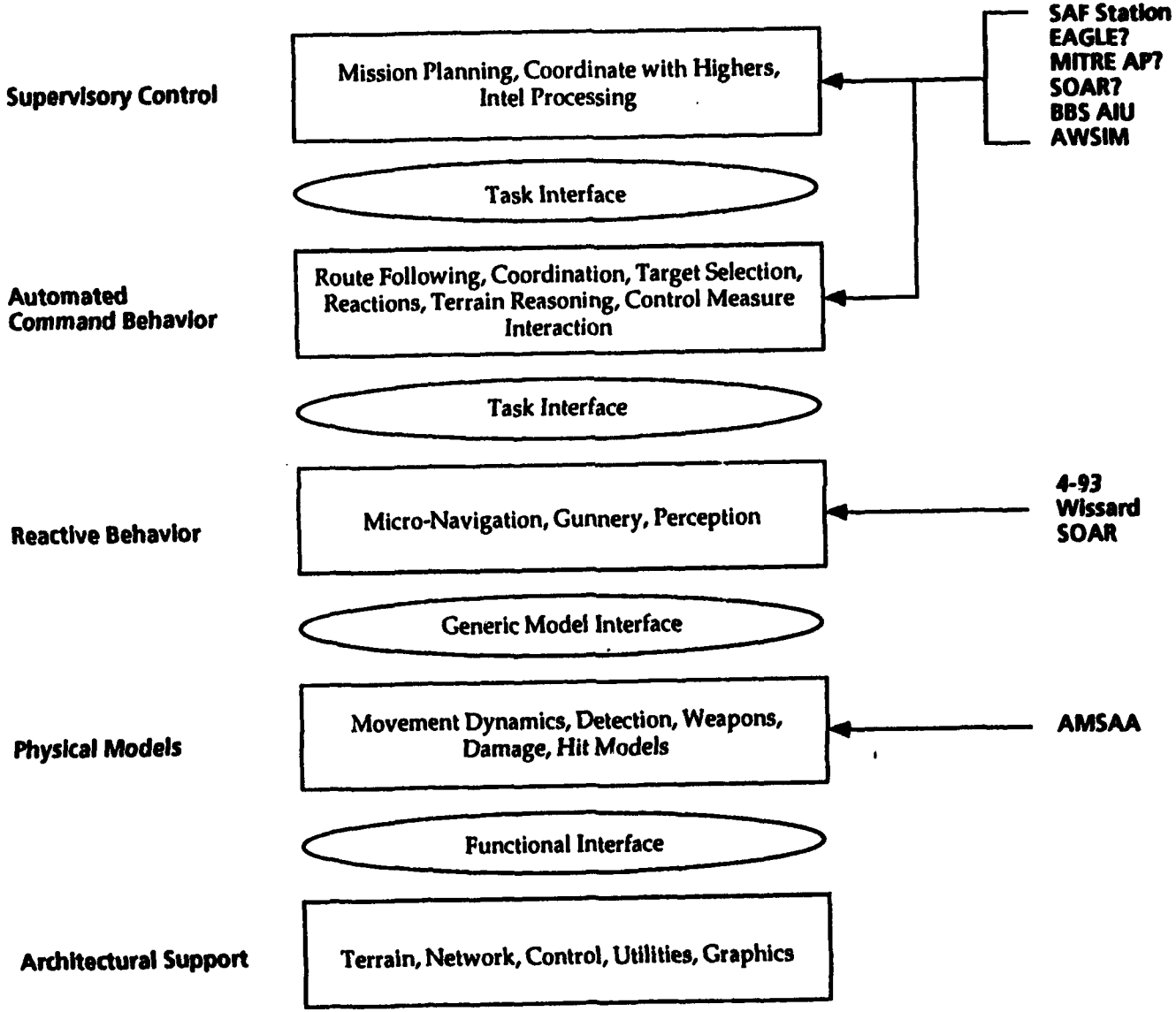
Architecture Design Objectives

- Support Arbitrarily Complex Missions (including preplanned contingencies and task reorganization)
- Frago Capability
- Override Choices made by Simulation
- Represent Missions Succinctly (like CIS)
- Provide General Framework for Encoding Behavior
- Allow Battlefield Uncertainty to be Incorporated
- Streamline User Interface Development
- Support Logging for Analysis and Restart
- Possible for User Interface to *Explain* Current Situation
- Language Independence

Architectural Framework

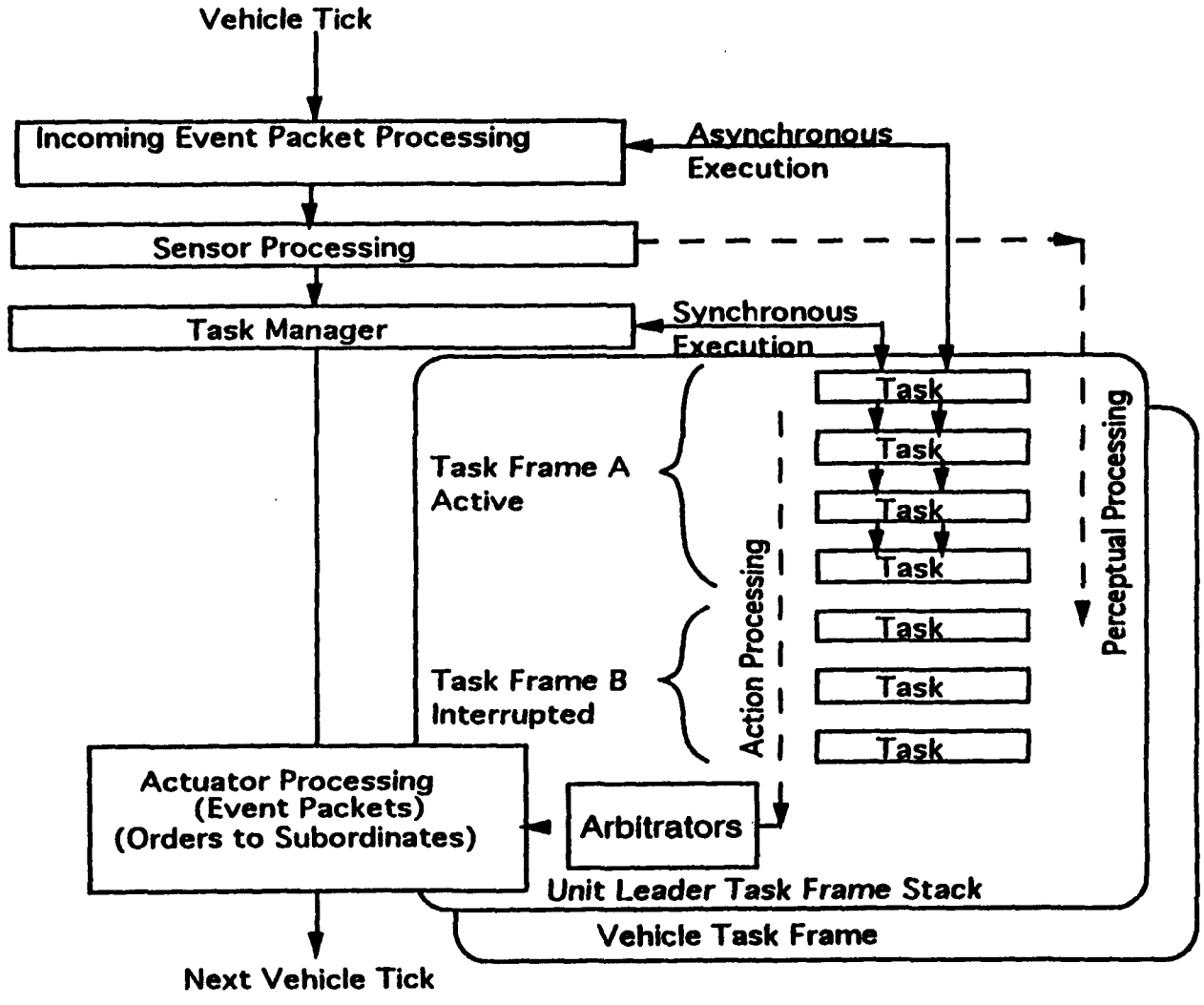
- **Task**
  - Unit Behavior or Individual System Control
  - Mission Specific or Always Present
  - Reflect Battlefield or Implement Simulation Details
- **Task Frame**
- **Unit Hierarchy**
- **Task Frame Stack**
- **Enabling Task**
- **Task Manager**

MODSAF LOGICAL ARCHITECTURE



LORAL
Advanced Distributed Simulation

Entity Simulation



LORAL
Advanced Distributed Simulation

Task Frame

- **Collection of Smultaneously Executing Tasks (similar to CIS)**
- **Built by...**
  - User Interface in Mission Construction
  - High-level Unit Tasks to Assign Sub-missions
  - Reactive Tasks to Push Reactions
- **Background Frame**
  - Unit Background Tasks
  - Vehicle Architectural Tasks
  - Holding Area for Assigned Tasks

Task Representation

- **Model Number**
  - Identifies Body of Software which Executes Task
- **Parameters**
  - Operating Parameters for a Task in a Particular Mission
- **State**
  - Used by Software Model at Runtime to Reflect Internal State

Unit Hierarchy

- **Dynamic Organizations**
  - Organic
  - Task
  - Functional
- **Unit Commanders**
- **Mission Assignment Algorithm**
  - Put unit ID in task frame (implies request to execute)
  - Unit commander *may* respond by executing task frame

Task Frame Stack

- **New Frame Pushed by Task to Implement Reaction**
- **New Frame Pushed by User to React to Surprises**
- **Transparent Frames**
  - Pre-programmed Instructions
  - Temporary Run Time Modifications

Enabling Tasks

- **Missions as a Sequence of Task Frames**
  - Enabling task is crucial task in previous frame
- **Missions as a Tree of Contingencies**
  - Enabling tasks implement predicates
- **Enabling Tasks can be Combined with Logical Operators**

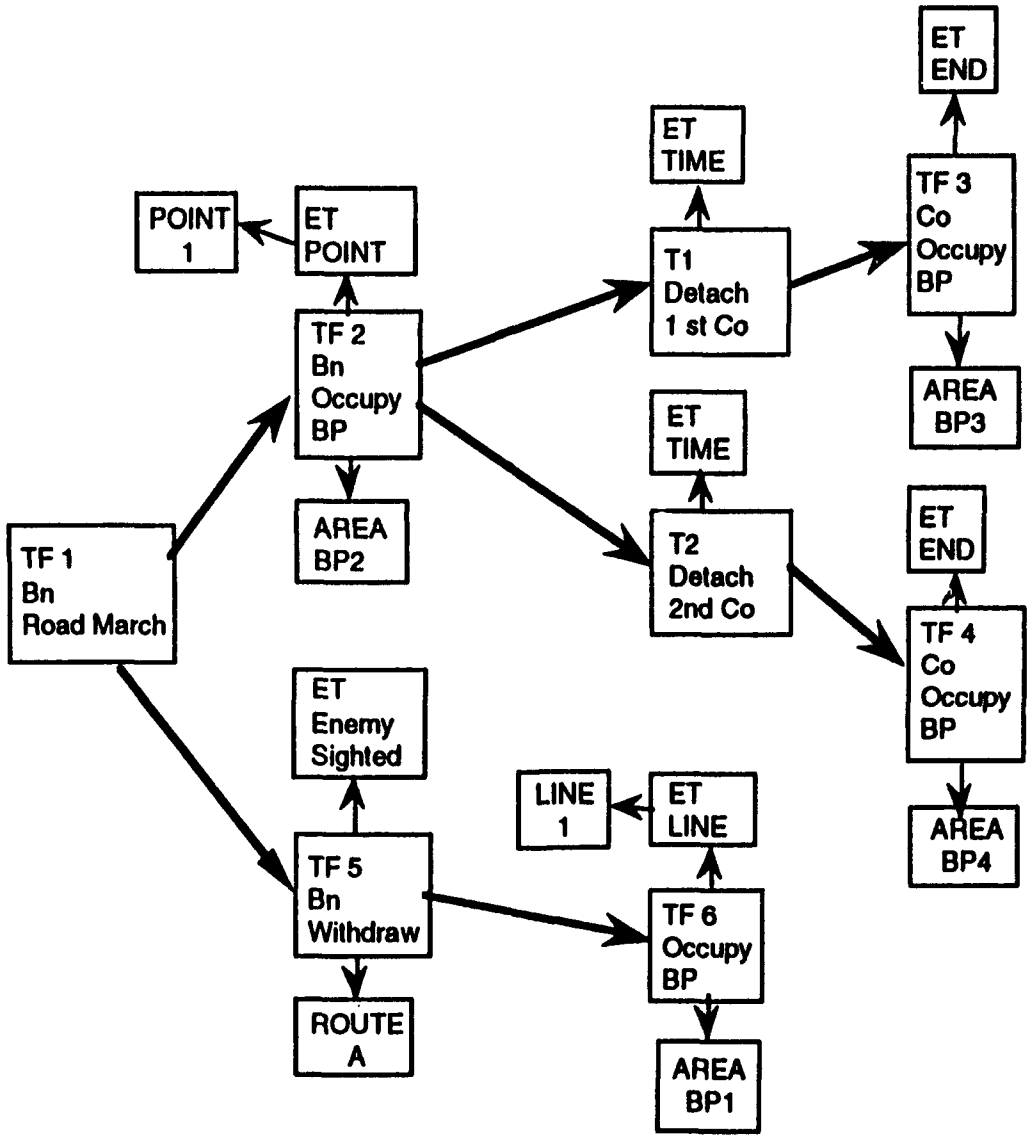
Task Manager

- **Registers Association between Model Number and Software**
- **Handles Task Inter-Dependency**
- **Invokes Tasks Synchronously**
  - Collect roles Vehicle is Playing
  - Checks for Unassignment
  - Check Enabling Tasks of Subsequent Frames
  - Traverse Current Frames of each Role and Assemble List of Tasks to Execute
  - Sort Execution List
  - Compare List to Last Tick, Suspend Missing Tasks
  - Execute Tasks In-order

ModSAF C2 Implementation

- **One Software Model (one library) Per Task**
- **Asynchronous Augmented Finite State Machines**
  - Built in TICK and PARAMS Events
  - Other Task Specific Events
  - Group Code by State rather than by Event
  - Automated State Transition Diagram Generation
  - Interface to Task Manager Generated Automatically
- **Arbitration**
  - Mutual, Context Free, Context Rich
  - Pull-based Information Flow
- **User Interface in Terms of Tasks, Task Frames**
  - Automated GUI Generation

Mission Example

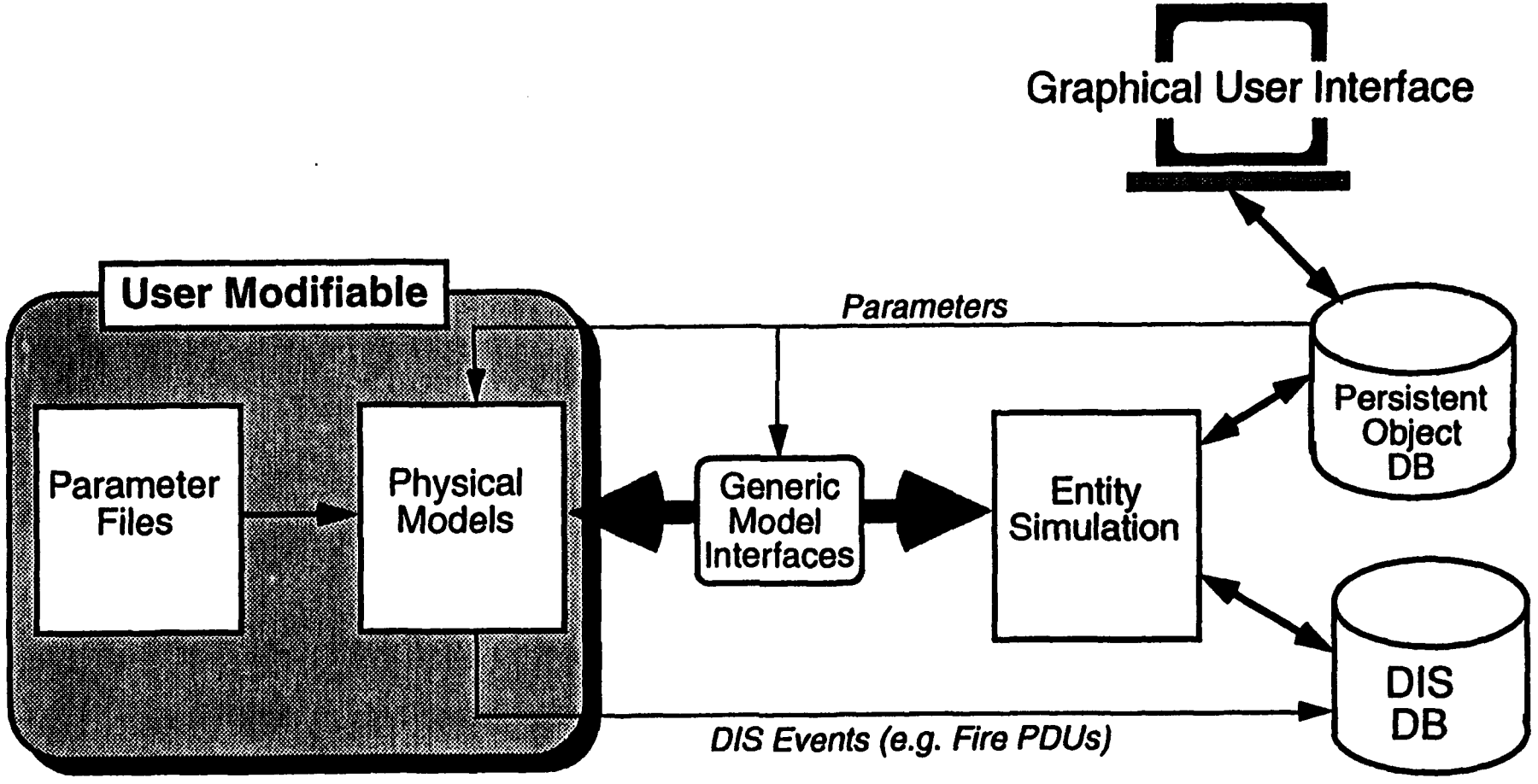


LORAL
Advanced Distributed Simulation

ModSAF Interfaces to Physical Models

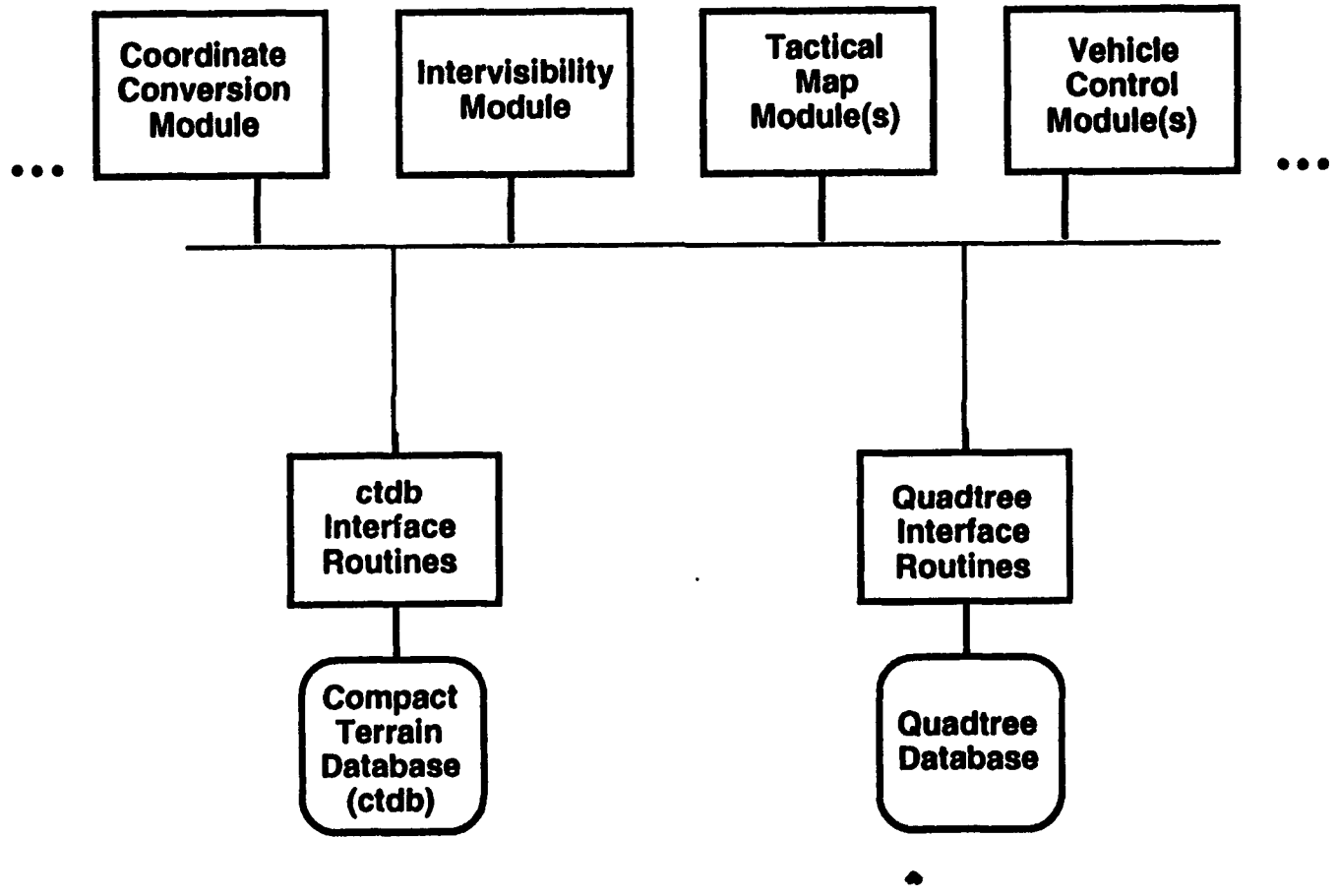
- Each Model is Parametric
- Models are Initialized from Parameter Files
- Model Parameters can be Changed Dynamically from the User Interface via Database
- Each Physical Model Persistent Object is Connected to the Simulation via a Generic Interface
- Changing the Parameters for the Generic Model Interfaces allows Physical Models to be Switched at Runtime
- This Capability can be used to Support Variable Resolution Simulation

ModSAF Interfaces to Physical Models



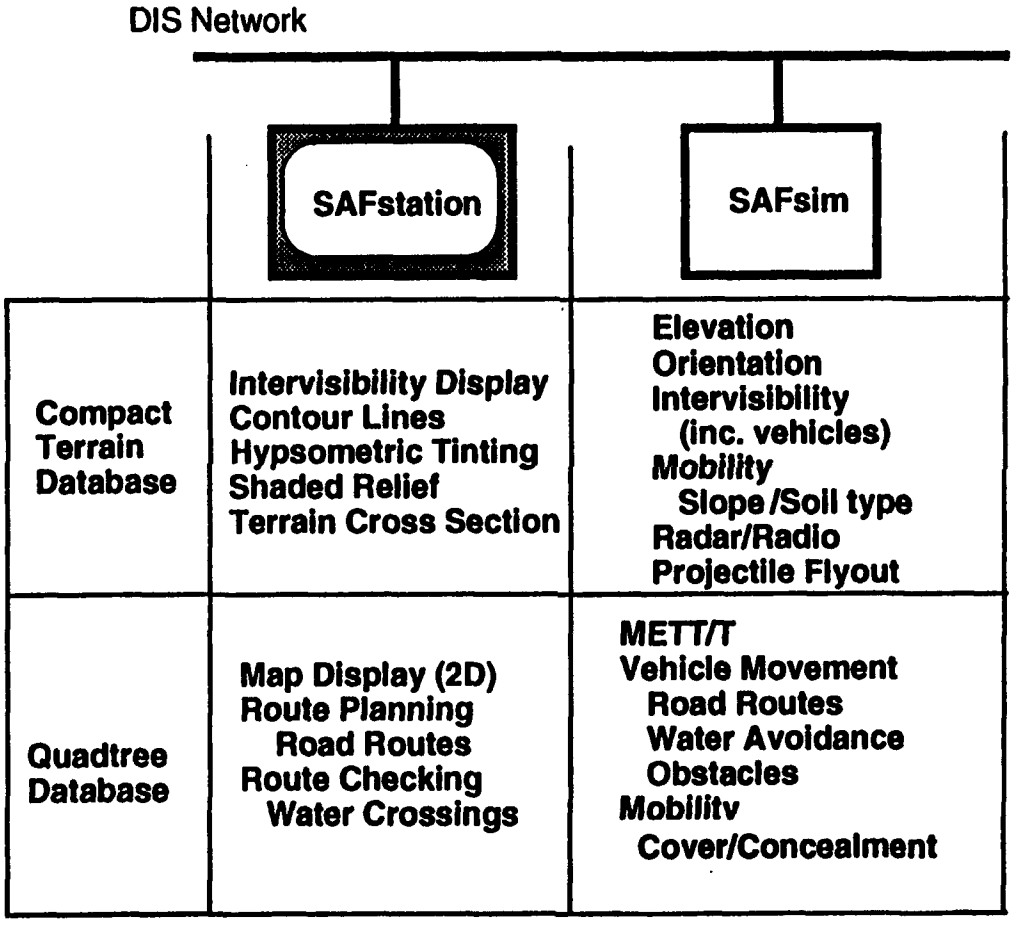
LORAL
Advanced Distributed Simulation

ModSAF Terrain Databases



LORAL
Advanced Distributed Simulation

ModSAF Component Terrain Usage



LORAL
Advanced Distributed Simulation

Compact Terrain Database (ctdb)

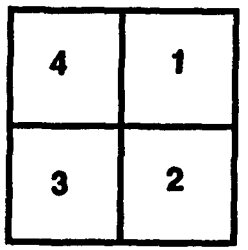
- Compression of Elevation Data by Exploiting Regular Nature of Terrain Database Modeling
- Microterrain &Terrain Features also Stored in Compact Format
- Multi-level Terrain Supported, for example, Multi-level Highway Bridges
- Geocentric or Flat Earth Terrain Models Supported
- Disk Caches for Elevation & Features Established at Runtime
- Separate Caches Used to Minimize Fragmentation
- Efficient Search Algorithms for Elevation Lookup & Intervisibility Calculations
- Optimized for RISC Processors

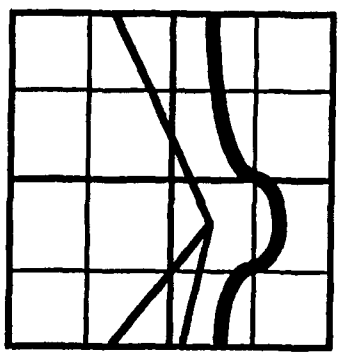
Compact Terrain Database

- **Elevation Grid**
- **Micro-terrain**
- **Buildings**
- **Tree and Tree Lines**
- **Canopy**
- **Linear** (roads and rivers)

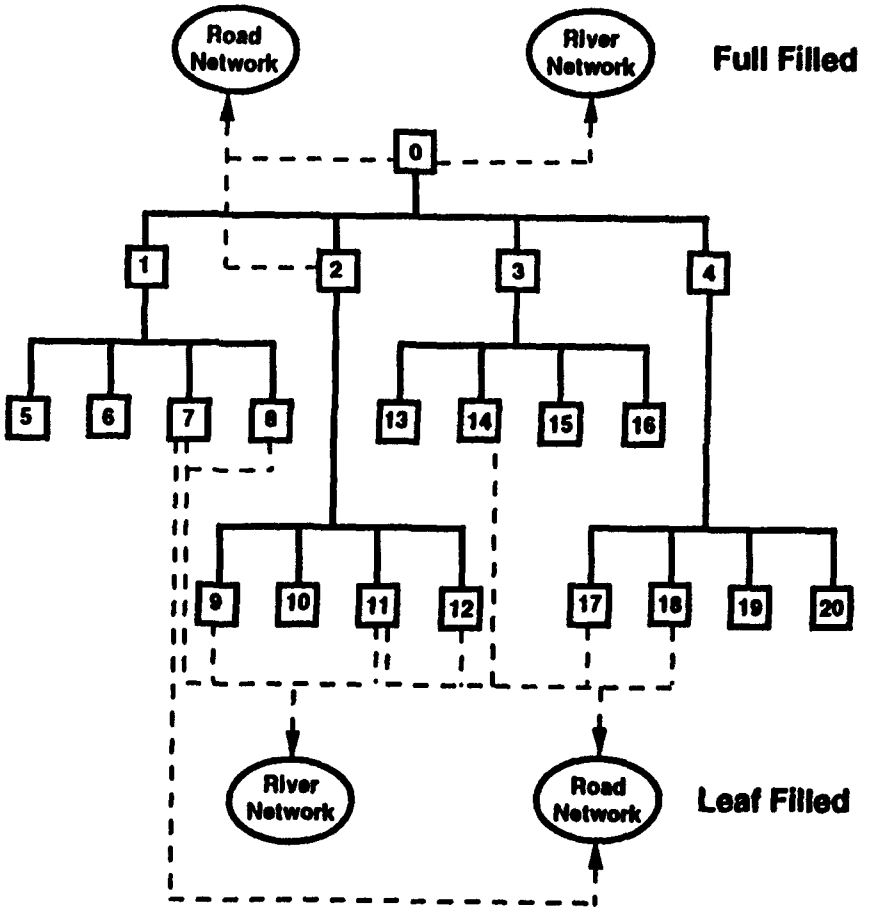
Quadtree Database

Quad Node Numbering



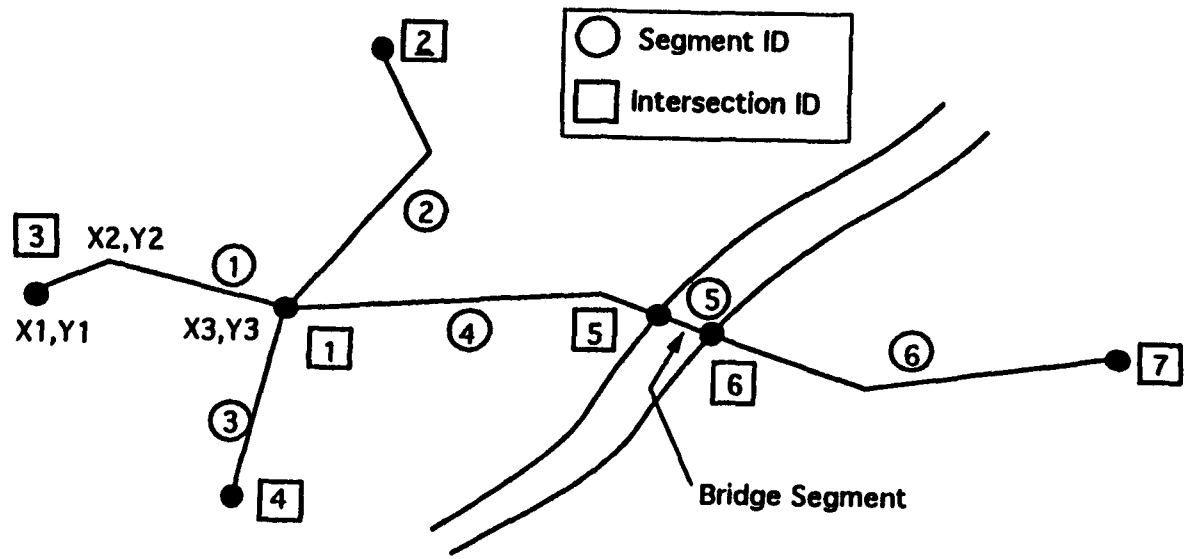


Terrain



LORAL
Advanced Distributed Simulation

Quadtree Feature Networks



Segment Array

0	Network Segment
1	Network Segment
2	Midline Points (X1 Y1 X2 Y2 X3 Y3)
3	Width
	Distance
	Other Attributes: Bridge (Road) Fordable (Water)

Intersection Array

0	Network Intersection
1	Network Intersection
2	Location (X3 Y3)
3	Segment ID 1 Intersection ID 3
	Segment ID 2 Intersection ID 2
	Segment ID 3 Intersection ID 4
	Segment ID 4 Intersection ID 5

Interfacing ModSAF with Other Systems

- **Interface to Physical Models via Generic Model Interfaces**
  - Hulls
  - Weapons
  - Sensors
- **Interface via Tasks and Task Frames Created and Monitored through PO Database**
- **Interface via Missions**

  Linked Task Frames Created by GUI or other Process through PO Database

ModSAF Versions

ModSAF A Preliminary Integration Version for IFOR
Released Dec 92

ModSAF B Air-to-Air Version
To be Released May 93; 15 sites

ModSAF C **Iniatial Ground Capability**
To be Released Sept 93

ModSAF 1.0 Full Air and Ground Capability
To be Released Nov 93

ModSAF 2.0 Advanced User Interface and Behavioral Capabilities

IFOR AI Based SAFOR Built on ModSAF Capable of Learning and Adapting

POINTS OF CONTACT

Government POCs

Name	Position	Telephone	E-mail
CDR Dennis McBride	Program Manager	703.696.2364	dmcbride@a.darpa.mil
Stan Goodman	STRICOM POC	407.380.8165	sgoodman@emh4-orlando.army.mil
Tom Tiernan	NRaD POC	619.553.3540	tiernan@cod.nosc.mil

LADS POCs

Name	Position	Telephone	E-mail
Andy Ceranowicz	Group Lead	617.873.3201	aceran
Rob Calder	ModSAF 1.0 PE	617.873.1875	rcalder
Josh Smith	Architecture	517.676.4044	jesmith
Barbara Perry	SAF Programs	617.873.3244	bperry
			@camb-lads.loral.com

LORAL
Advanced Distributed Simulation

ACRONYMS/DEFINITIONS

73 Easting	Battle in Desert Storm
AGPT	German DIS Platoon Trainer
AI	Artifical Intelligence
AIU	Advanced Interface Unit
AMSAA	Army Materiel Systems Analysis Activity
AP	Adversarial Planner
AWSIM	Air War Simulation
BBS	Brigade/Battalion Battle Simulation
BN	Battalion
BP	Battle Position
BVR ATAC	Beyond Visual Range Air-to-Air Combat
C2	Command and Control
CIS	Combat Instruction Set
CO	Company

ACRONYMS/DEFINITIONS

CSS	Combat Service Support
CTDB	Compact Terrain Database
CVCC	Combat Vehicle Command and Control
DB	Data Base
DI	Dismounted Infantry
DIS	Distributed Interactive Simulation
Entity	Any *SAF* model; e.g., vehicle, dismounted infantry
ET	Enabling Task
FAADS	Forward Area Air Defense System
Frago	Fragmentany Order
FS	Fire Support
GUI	Graphical User Interface
IFOR	Intelligent Forces

ACRONYMS/DEFINITIONS

Mission	Network of *Task Frames* linked together to form an execution tree
ModSAF	Modular Semi-Automated Forces
ODIN	DARPA Project on Visualization of Intelligence data
PDU	Protocol Data Unit
PO	Persistent Object database-captures shared command and control information e.g., units, messages, missions
POP	Persistent Object Protocol
PVD	Plan View Display
Reactions	*Task Frames* that deal with changes in the battlefield environment
RISC	Reduced Instruction Set Computer
SAF	Semi-Automated Forces
SAF-logger	ModSAF data logger that records the exercise and allows replay and restart
SAFsim	SAF Simulator

ACRONYMS/DEFINITIONS

SAFstation	SAF user interface workstation
Scenario File	Contains information required to initialize a simulation exercise
SGI	Silicon Graphics Incorporated
SIMNET	DARPA SIMulator NETworking project
SOAR	AI system for simulating cognition of agents
Task	A behavior performed by a ModSAF entity or unit on the battlefield
Task Frame	Collection of related tasks that run concurrently
TF	Task Frame
V&V	Validation and Verification

APPENDIX E

COMPUTER GENERATED FORCES BRIEFING (INSTITUTE FOR SIMULATION AND TRAINING)

Institute for Simulation and Training

Computer Generated Forces

IST's CGF Research Projects

Intelligent Simulated Forces (ISF)
Basic research in CGF behavior generation algorithms.
N61339-92-C-0045
Sponsor: STRICOM; COTR: Paulson; PI: Petty

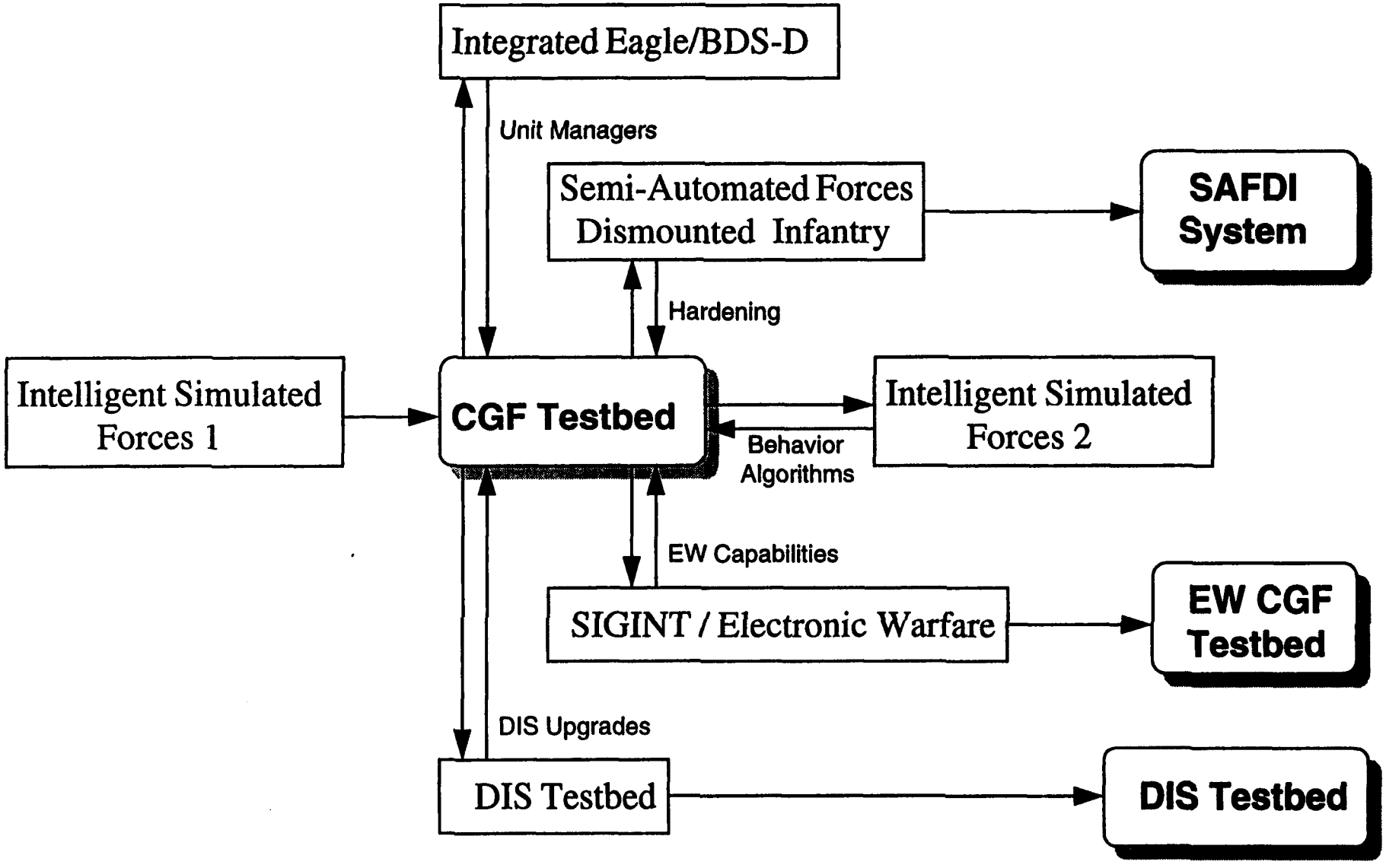
Semi-Automated Forces Dismounted Infantry (SAFDI)
CGF development focused on Dismounted Infantry capabilities.
N61339-93-C-0026
Sponsor: STRICOM; COTR: Paulson; PI (Acting): Franceschini

Integrated Eagle/BDS-D (Eagle/BDS-D)
Linking a "constructive" aggregate level wargame (Eagle) with a "virtual" vehicle level simulation (SIMNET/BDS-D/DIS).
N61339-92-K-0002
Sponsor: STRICOM; COTR: Paulson; PI: Karr

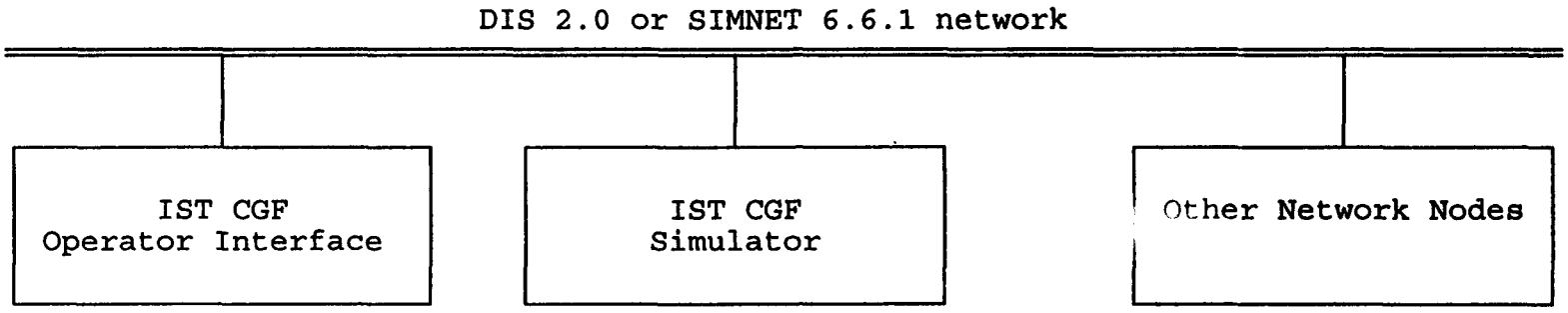
SIGINT/Electronic Warfare (EW)
Standards, databases, and CGF software for Electronic Warfare.
N61339-91-C-0091 (ECP to DIS Standards)
Sponsor: STRICOM; COTR: Williams; PI: Wood

Agenda

Topic	Presenter	Duration
Overview of IST's CGF research	Petty	:10
SAFDI	Petty	:15
Integrated Eagle/BDS-D	Petty	:15
Intelligent Simulated Forces		
Overview of ISF	Petty	:05
Terrain reasoning	Petty	:15
CGF C-to-Ada conversion	Craft	:45


Integrated Eagle/BDS-D
Unit Managers
Semi-Automated Forces
Dismounted Infantry
SAFDI System
Hardening
Intelligent Simulated Forces 1
CGF Testbed
Intelligent Simulated Forces 2
Behavior Algorithms
EW Capabilities
SIGINT / Electronic Warfare
EW CGF Testbed
DIS Upgrades
DIS Testbed
DIS Testbed

IST CGF Testbed Components



Simulator

- Performs "simulation" of CGF entities:
  - vehicle dynamics
  - dead reckoning
  - Line of Sight determination
  - damage assessment
- Behavior generation

Operator Interface

- Used by CGF Operator to command the CGF entities
- Mouse and keyboard input, plan view display output
- OI and Simulator communicate via main network

IST CGF Testbed Characteristics

Hardware platform
- 2x IBM-compatible ISA Personal Computers (OI and Simulator)
- 486DX2 33/66 CPU, 4+M RAM, VGA, 3Com Etherlink II

Code
- ANSI C, compiled with C++ compiler (Borland C++)
- Simulator: 58,000 lines, Operator Interface: 40,000 lines

Behavior generation
- Finite State Machines used for both time-slicing and behavior partitioning
- Behaviors have been constructed bottom up

Semi-Automated Forces Dismounted Infantry

N61339-93-C-0026

Why develop SAFDI?

"Real battlefields have dismounted infantry; SIMNET doesn't."

Without dismounted infantry, SIMNET (or DIS) provides:

An unrealistic training environment.
- M1 crews move without concern for hidden ATGM-armed infantry
- M2 crews fight mounted only, without learning when to dismount

An incomplete tool for analysis and development.
- New tactics and weapons systems can't be tested against infantry
- Infantry tactics and weapons systems can't be tested at all

No opportunity to learn about simulating dismounting infantry.
- Lessons learned in SIMNET about vehicle simulation benefit CCTT
- No equivalent opportunity to learn about simulating infantry
- CCTT and other future simulations require dismounted infantry

Because of scarcity and lack of realism, the Dismounted Infantry Module (DIM) does not meet these needs.

Goals and deliverables of the SAFDI project

Goals.

- Provide a stable dismounted infantry system for evaluation at SIMNET/BDS-D sites.
- Develop dismounted infantry capabilities in a CGF system.
- Build up experience in using dismounted infantry in a DIS-type simulation.

Deliverables.

- Basic SAFDI system (August 1993).
- Enhanced SAFDI system (February 1994).

User evaluation of SAFDI

The SAFDI is a hardened system suitable for user evaluation at training sites, but it is not a training system.

User evaluation of SAFDI may answer several important questions.

- Is the SIMNET/DIS training environment improved by dismounted infantry? Why and to what extent?
- Should an effort be made to train infantry officers and NCOs with a SAFDI system, or should the emphasis be on vehicle crews?
- How can the SAFDI system be improved, for future research or for fielding as a training device?
- What aspects of dismounted infantry simulation are important or problematic for future simulations such as CCTT?

Effective evaluation of SAFDI will require its use in training-like exercises.

SAFDI entities

SAFDI fireteam.
- 1 ATGM gunner, 1 SAW gunner, 1 grenadier, 2 rifleman
- US ATGM: Dragon
- OPFOR ATGM: RPG16

SAFDI infantry fighting vehicles.
- US IFV: M2; 25mm main gun, coaxial MG, TOW
- OPFOR IFV: BMP-2; 30mm main gun, coaxial MG, Spandrel

SAFDI main battle tanks.
- US MBT: M1; 105mm main gun, coaxial MG
- OPFOR MBT: T-72; 120mm main gun, coaxial MG

Basic SAFDI capabilities: Movement

Move, change speed, change posture.

- Movement ordered by operator, route planned by software
- Route planner avoids obstacles, accepts waypoints
- Speed set by operator (Stop, Normal, Double, Rush)
- Postures set by operator or by action (Standing, Kneeling, Prone)
- Automatic postures changes (e.g. kneel when firing ATGM)

Energy and fatigue modeled for fireteams.

- Energy level maintained for each fireteam
- Movement at Double and Rush speeds reduces energy
- Exhausted fireteams slow down automatically
- Resting increases energy

Mount and dismount vehicles.

- Fireteams can mount friendly IFVs and helicopters
- Mounted vehicles may be SAFDI, manned simulators, BBN SAF
- No more than one SAFDI fireteam can mount a single vehicle

Basic SAFDI capabilities: Combat

Kill enemy infantry and vehicles.

- Rules of engagement specify when to fire
  - Fire within range
  - Fire when
- Target chosen automatically according to target priorities
- Weapon chosen automatically depending on target and range
- Ammunition expenditure and resupply tracked
- Self-preservation range

Be killed by enemy fire.

- Vulnerable to direct and indirect fire
- Vehicles suffer standard SIMNET damage levels
- Fireteams suffer incremental losses
- Automatic weapon redistribution within fireteam

Be suppressed by enemy fire

- Incoming fire may also cause suppression
- Probability set in configuration file
- Suppressed fireteam loses control of ATGM in flight
- Suppressed fireteam may not fire

Basic SAFDI capabilities: Communication

Report activity within Line of Sight.

- Line of sight determinations performed automatically
- Sighting model considers sighter's ability to see, target's noticability
- Sighting results: detect, recognize, identify
- Sighting model parameters in configuration file
- Sightings reported to SAFDI operator in message pane

Communicate with higher headquarters.

- SAFDI operator is 'higher headquarters'
- Entities accept commands from operator
- Entities report status and sightings to operator

Basic SAFDI capabilities: Group commands

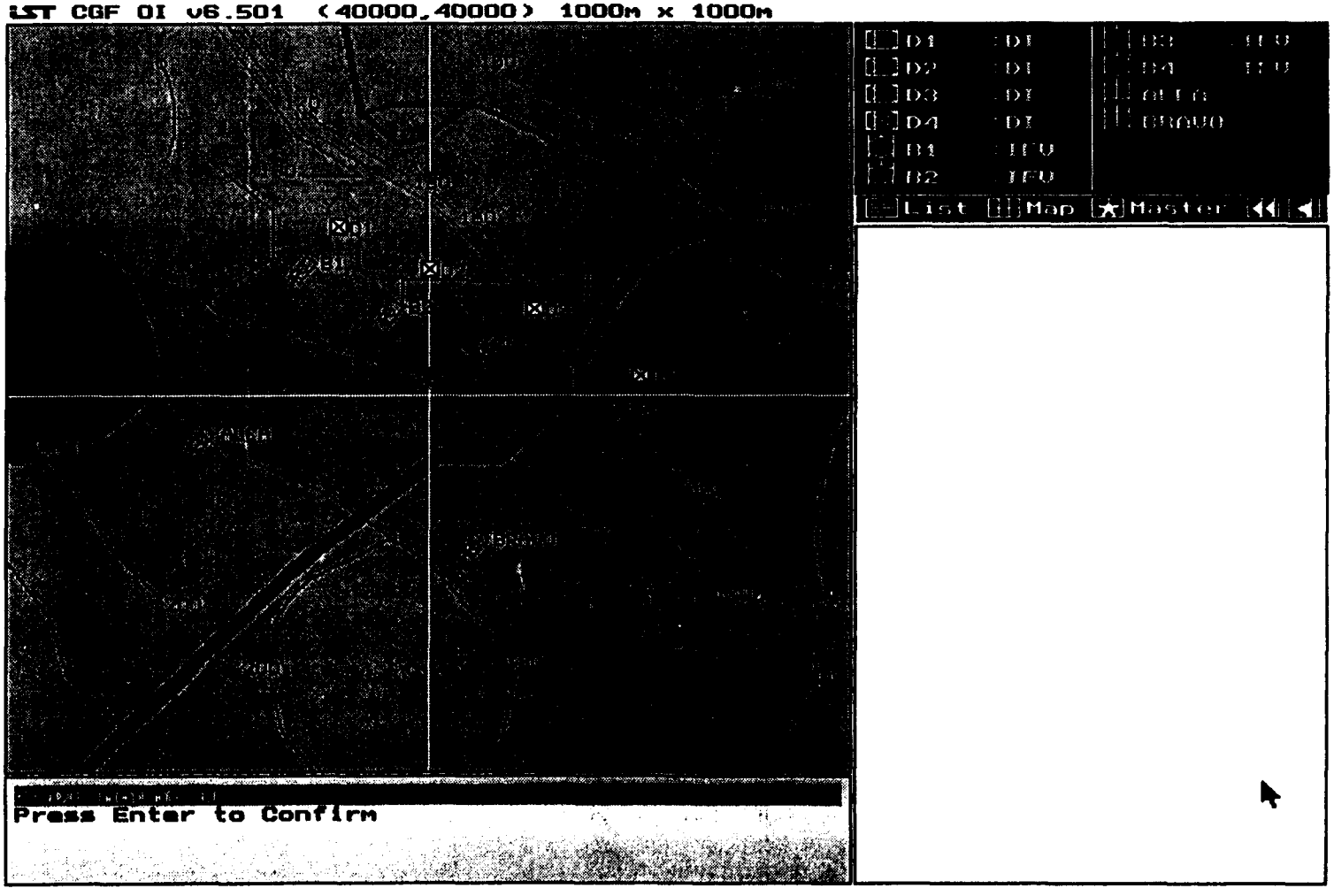
Create hierarchy of control.

- Controlled entities may be assigned to a group
- Groups may be assigned to higher level groups
- Groups can be organized like platoons and companies

SAFDI operator can give orders to groups.

- Issue order once to group, passed along to subordinates
- Some orders are 'interpreted' to make sense (e.g. Mount, GoTo)
- Groups are primarily 'couriers' now, but will be 'planners' later
- Special group 'ALL' includes all SAFDI entities on the SAFDI station

Group Commands
Example: Mounting



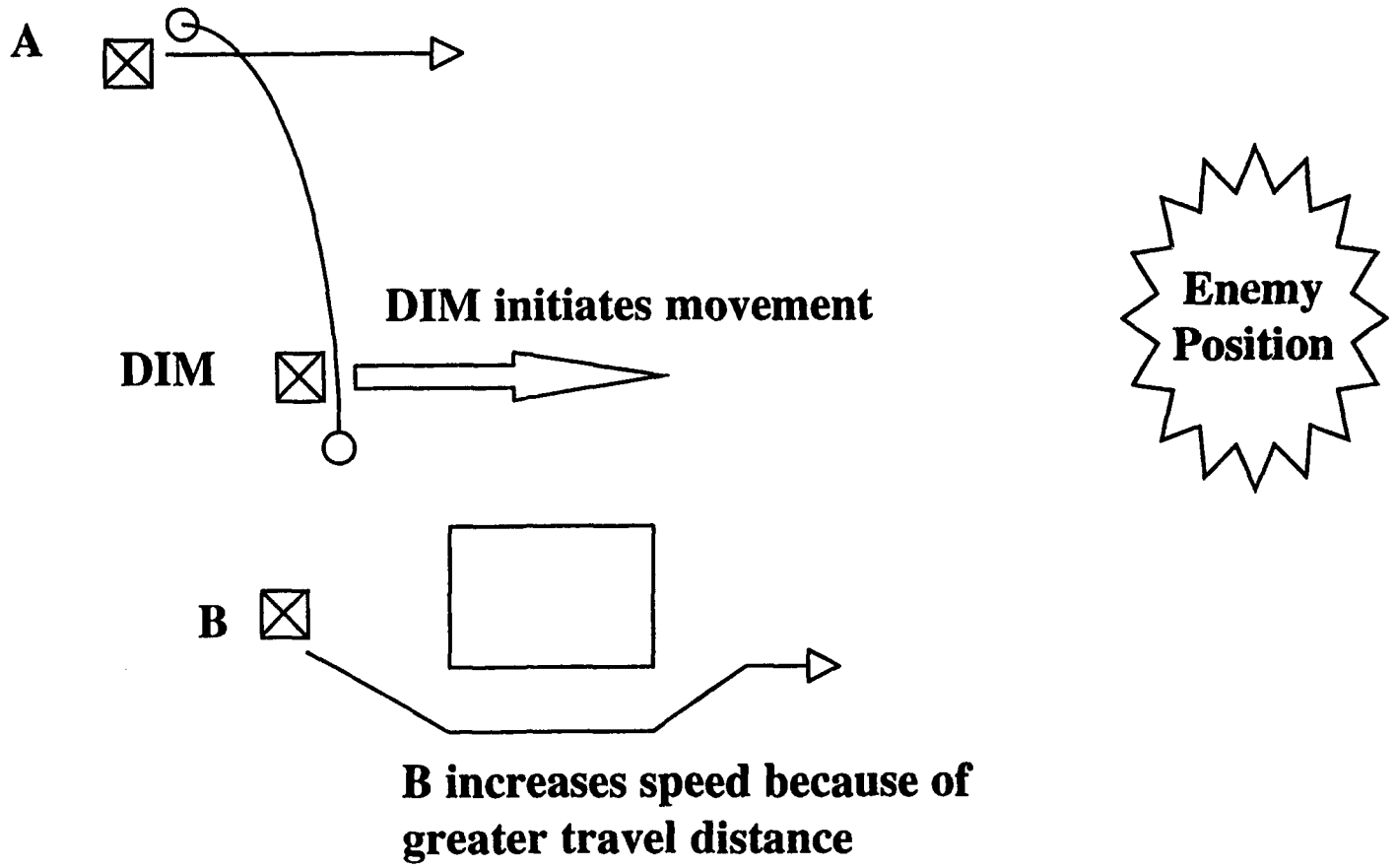
Basic SAFDI capabilities: Attach and Follow

SAFDI entities or groups can be attached to follow other entities.
- Other SAFDI entities
- Manned simulators (M1, M2, DIM)
- BBN SAF entities

Followers move along with leader.

SAFDI MBTs and IFVs have a 'Walk' speed setting.
- Matches DI fireteams' Normal movement speed
- Allows vehicles to lead or be led by fireteams

Attach and Follow
DIM leads assault



Basic SAFDI capabilities: Configuration files

Many SAFDI system parameters are stored in configuration files.

- Mount and dismount delays
- Ph tables
- Pk tables
- Missile dynamics (start speed, maximum speed, maximum turn)
- Suppression probability and recovery time
- Vehicle characteristics (size, speed, acceleration, weapons)
- Line of Sight interval
- Visibility and sighting parameters

The configuration files are plain text files.

- Modifiable with any text editor, such as DOS edit
- File format designed to be easy-to-modify by site support personnel
- Parameter values are completely documented in the SAFDI Support Manual

SAFDI documentation

SAFDI User's Guide.

- Introduction to SAFDI, should be read by everyone
- Step-by-step tutorial on use of the system
- Intended for use in training SAFDI operators
- Does not require computer knowledge

SAFDI Support Manual.

- Description of system installation procedures
- Troubleshooting instructions
- Detailed documentation of configuration file parameters
- Discussion of possible uses of SAFDI in training-like situations
- Intended for site support personnel and trainers
- Computer and SIMNET background required

Both will be updated with each revision of the software.

Functionality testing

Planned testing.
- Set of test cases with clearly defined expected results and procedures
- Designed to execute all SAFDI commands and functions, in all variations

Scenario testing.
- Testers 'fought' SAFDI vs. SAFDI and SAFDI vs. BBN SAF battles with military objectives
- Testers and observers watched for system errors

Source code coverage.
- Walkthroughs and inspection of source code by programmers
- Special purpose debugging code written to exercise other code segments

Site testing.
- SAFDI testing conducted at Ft. Benning and Ft. Stewart
- Interface of SAFDI with existing SIMNET equipment

Examples of difficult functionality tests

To test stability of the Follow command:
Follow command for 50+ Km, with many obstacles and waypoints.

To stress the route planner:
GoTo command with route containing 2500 waypoints in convoluted order.

To test target prioritization routines:
Permission to fire given with 80+ targets available.

To test for rare or slow-developing problems:
Long duration scenarios (1 hour to 8 hours).

Testing results

Planned test cases: approximately 1300.

Functionality defects found: approximately 200.

Defects corrected: approximately 170.

Examples of uncorrected defects:

- More than 32,000 missiles fired in single exercise causes SIMNET compliance error
- DI fireteam reports 'Lost sighting' upon mounting
- DI fireteam mounted on carrier entity that is removed from the scenario, e.g. manned simulator restart, becomes 'undead'

Network load capacity testing

Tests of the SAFDI system's network load capacity were based on scenario tests.

Load testing scenario size.
- 2 SAFDI stations with 12 internal entities each
- 80+ external BBN SAF or MCC vehicles

Scenario activities.
- SAFDI entities move back and forth through external entities
  - many Line of Sight checks
  - route planner must route around vehicles
- Permission to fire given to SAFDI, then OPFOR
  - difficult target prioritization
  - simultaneous missile and gun combat

Other network load.
- Additional entities on other exercise IDs
- Non-SIMNET traffic on network (Internet or sendfile/getfile utilities)

Network load capacity

Rated capacity per SAFDI station.
- 12 internal on SAFDI station + 40 external on network
- Promised and supported by IST
- Based on 'worst case' situations

Estimated actual capacity.
- 12 internal + 60 external within 3 Km + 60 external beyond 3 Km
- 3 Km is 'sighting range'
- Total of 120 external

'Red flag' mode invoked under extreme network load stress.
- During temporary system overload periods, non-critical PDUs are ignored
- Load must reach approximately 3-4 times rated capacity before significant number of Impact PDUs are lost

The SAFDI Support Manual discusses network load.
- Network overload symptoms in the OI and Simulator
- Techniques to avoid overload

SIMNET problems related to SAFDI: Manned simulators

Manned simulators have no coaxial machine gun capabilities.

- Dismounted infantry can not be attacked with preferred weapon
- Other means are currently available
  - Main gun fire
  - Collisions
  - Indirect fire request
  - Engage hostile DI with friendly DI
- Possible future fixes
  - Retrofit coaxial MG capability to simulators
  - SAFDI system fires coaxial MG for vehicles

Mounting SAFDI fireteams on manned M2 simulators requires coordination.

- Radio link between M2 and SAFDI operator needed
- M2 must remain motionless during mount process

SAFDI Delivery and Installation

Delivery

- Forts Benning and Stewart, week of 13 September (on time)
- Delivery process
  1. Installation
  2. Overview Presentation
  3. Capabilities Demonstration
  4. Operator Training
  5. Support Training

Evaluation

- Fort Benning, 26-29 September
- Three training scenarios

Evaluation Scenarios

The Basic SAFDI system participated in actual training of A/1-29 Infantry (CAPT William Hessenius, Commanding).

- Scenario #1: Defense
- Scenario #2: Offense
- Scenario #3: Take Crash Hill

Scenario #1: Defense

US A Company	**OPFOR Motorized Rifle Battalion**
8x M2 (Manned Simulator)	40x BMP (BBN SAF)
4x M1 (Manned Simulator)	16x T-72 (BBN SAF)
8x DI fireteam (SAFDI)	6x DI fireteam (SAFDI)
12x SPA (MCC)	2x Su-25 (BBN SAF)
	2x Mi-24 (BBN SAF)
	12x SPA (MCC)

- A Company defends against a frontal assault by the MRB.
- 2 SAFDI platoons (8 fireteams) dismounted in defense.
- US SAFDI fireteams destroyed OPFOR vehicles with Dragon.
- OPFOR SAFDI fireteams conducted dismounted assault on southernmost M2 platoon, destroying three manned M2s with RPG16 fire.

Scenario #2: Offense

US A Company	**OPFOR MRP+**
8x M2 (Manned Simulator)	4x BMP (BBN SAF)
4x M1 (Manned Simulator)	1x T-72 (BBN SAF)
8x DI fireteam (SAFDI)	4x DI fireteam (SAFDI)
2x OH-58 (BBN SAF)	2x Mi-24 (BBN SAF)
12x SPA (MCC)	12x SPA (MCC)

- A Company forges through two narrow passes defended by OPFOR.
- US SAFDI fireteams transported by M2s dismount and conduct dismounted assault; they clear the first pass of T-72s, BMPs, and OPFOR SAFDI fireteams.

Scenario #3: Take Crash Hill

US A Company	**OPFOR MRC**
8x M2 (Manned Simulator)	6x BMP (BBN SAF)
4x M1 (Manned Simulator)	5x T-72 (BBN SAF)
8x DI fireteam (SAFDI)	6x DI fireteam (SAFDI)
12x SPA (MCC)	12x SPA (MCC)

- A Company encounters OPFOR MRC ambush while attempting to reach Crash Hill before OPFOR MRB arrives.
- OPFOR SAFDI fireteams aid in forward ambush of US forces.
- US SAFDI fireteams dismount and destroy BMPs and OPFOR DI during forward ambush.

Evaluation Observations (IST's perspective)

- The SAFDI system was completely reliable throughout evaluation; it ran for a total of 16 hours without crashing even once.
- Scenario #1 contained 120+ entities--3 times rated capacity.
- Site's production SAF system crashed 3 separate times during the evaluation exercises.
- With DI, pace of battle slowed (35 min. scenario took 2 hour).
- SAFDI entities participated realistically in the battles.
- SAFDI system was operated by evaluators, not IST personnel.

Evaluators' Comments (from a draft of the Evaluation Report)

- CAPT William Hessenius, CO A/1-29
  "... **SAFDI greatly increased my unit's training.** It expanded the SIMNET battlefield and allowed myself the unit commander to assess my unit in areas not realized before in SIMNET." (Emphasis added.)

- John D'Errico, Dismounted Warfighting Battle Lab
  "SAFDI now presents the mechanized leaders with the requirements to to consider the infantry presence ..."
  "A total of 93% of the soldiers who participated in the SIMNET/SAFDI exercises stated that they would use SAFDI again, and preferred training in SIMNET with SAFDI added ..."

Enhanced SAFDI

Enhanced SAFDI to be delivered to Ft. Benning and Ft. Stewart in February 1994.

Changes and corrections identified by the evaluation, to the extent possible.

Considerable new functionality.

- On-line help for Operator Interface
- Feasibility of additional DI icons
- Indirect fire generation and Forward Observers
- Air defense weapons (Stinger and SA-7 Grail)
- Minor realism enhancements
  - track casualties to individuals in the fireteam
  - parameterized fireteam makeup
  - consider fireteam exhaustion in Ph
- System stress warnings

Updated documentation.

Integrated Eagle/BDS-D

N61339-92-K-0002

Goal:

- Demonstrate the feasibility of the interoperability of simulations supporting different entity granularities and different time scales.

Specific Accomplishment:

- Develop a mechanism for integrating the operation of an aggregate combat simulation (Eagle) with a vehicle simulation (IST Computer Generated Force (CGF) Testbed operating in SIMNET).

Organizations Involved:

- TRAC - developer of the Eagle simulation.
- Los Alamos National Laboratory (LANL) - to develop Eagle/SIMNET interface.
- Institute for Simulation and Training (IST) - developer of IST CGF Testbed.

Eagle Simulation

- Corps/division level combat model developed at TRAC.
- Smallest units (maneuver units): battalion and company.
- Used in analysis of the effects of:
  - weapons systems
  - command and control
  - military doctrine
  - organization.

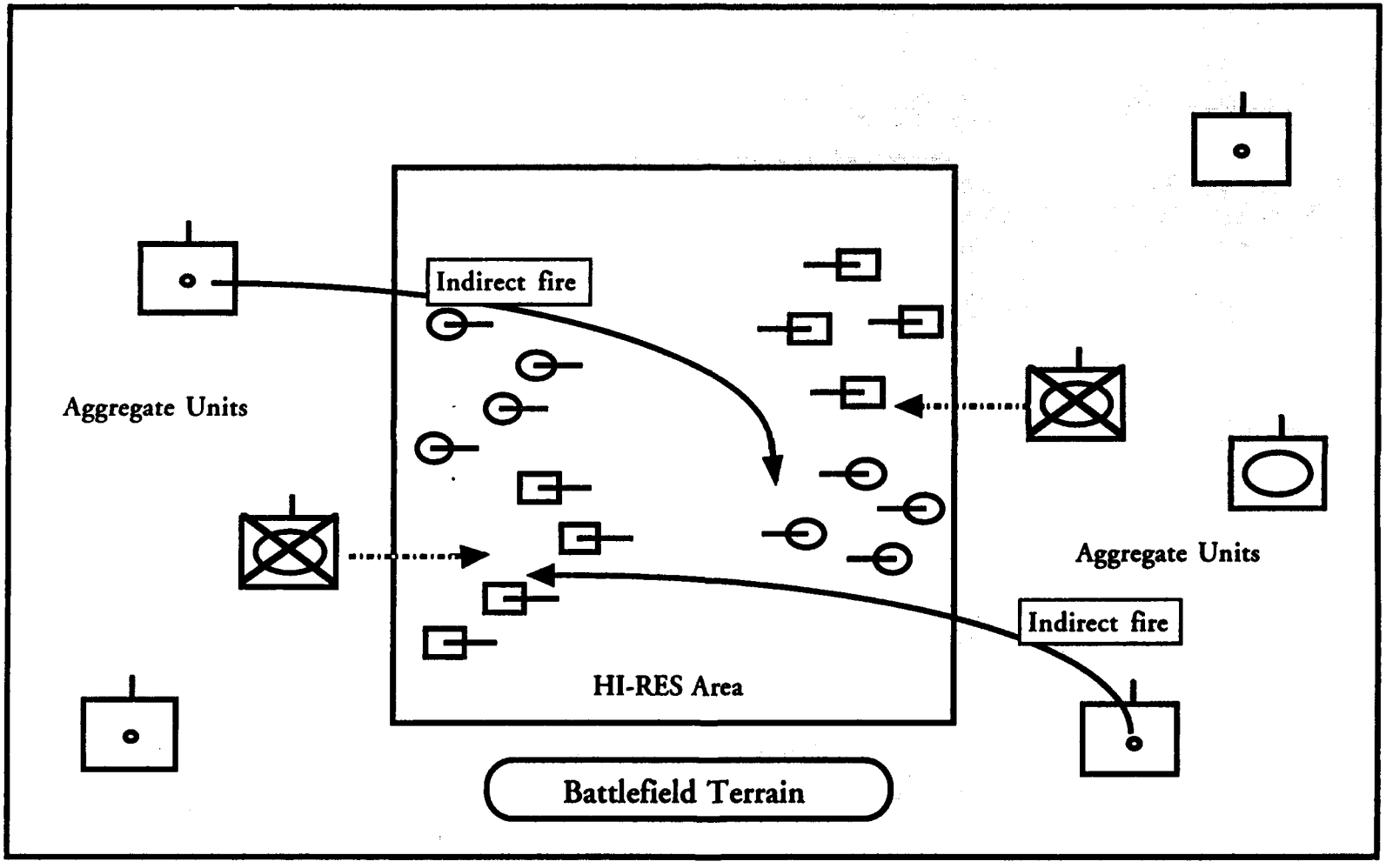
BDS-D (SIMNET)

- Well known distributed interactive simulation system.
  - individual vehicle simulators interact in a shared environment using a computer network.
- Used in training tank crews in cooperative team tactics.

IST CGF Testbed

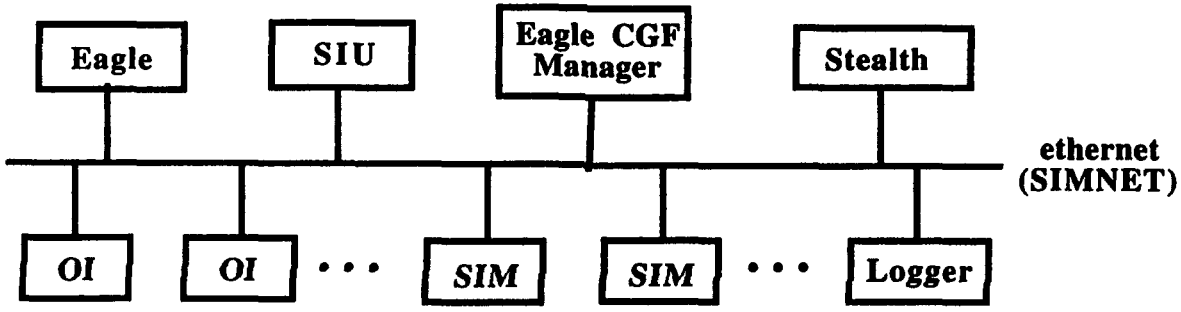
- SIMNET-compatible Computer Generated Force (CGF) system a.k.a. Semi-Automated Force (SAFOR).
- Used in research of CGF techniques and algorithms.

Typical Eagle/BDS-D Scenario



Integrated Eagle/BDS-D

Network Configuration



Eagle	- TRAC Eagle Simulation
SIU	- Simulation Integration Unit
Eagle CGF Manager	- IST CGF Testbed Manager
OI	- IST CGF Testbed Operator Interface
SIM	- IST CGF Testbed Simulator
Logger	- IST CGF Testbed Logger
Stealth	- NPS personal Stealth

Results to Date

- Original Proof of Concept Functionality
  - Disaggregation of aggregate units into vehicles in virtual world.
  - Reaggregation of virtual vehicles into aggregate unit.
  - Operational Orders and Operator Intents.
    Communicate Op Orders controlling Eagle unit to CGF operator and operators' intent to unit commanders in Eagle.
  - Indirect Fire from Eagle world into virtual world.
    CGF operator can request artillery support from aggregate Eagle units. Indirect Fire allocated within Eagle according to the requirements of the larger Corps/division/brigade battle.

- Recent Functionality
  - Track Eagle units within CGF Testbed.
    Aggregate icons visible on Operator Interface.
  - Base Eagle unit detection on BDS-D Line of Sight (LOS) sightings.
    Detections between disaggregated units based on vehicle LOS.
  - Battalion and Platoon Disaggregation
    Battalion, company, and platoon disaggregations.
  - Include Manned Simulators in Disaggregation.
    Manned simulators can be components of disaggregated units.

Disaggregation ≠ Deaggregation (DIS)

Aggregation and Deaggregation (DIS)

- Initially, individual vehicles exist in simulation.
- Individual vehicles are aggregated to reduce network traffic.
- When other entities must "see" the vehicles, the aggregate is deaggregated.

Disaggregation and Reaggregation

- Initially, aggregate units are simulated; their individual vehicles do not "exist".
- Aggregate units can be disaggregated; i.e. individual vehicles are created.
- Later, the disaggregated unit may be reaggregated.

Summary

Aggregation (DIS) - groups individual vehicles to reduce traffic.
Deaggregation (DIS) - reveals hidden vehicles.

Disaggregation - allocates simulation resources to create individual vehicles.
Reaggregation - removes individual vehicles and releases simulation resources.

How they work together

An aggregate simulation is operating in an exercise. A sensor platform (e.g. JSTARS) requests that an aggregate unit deaggregate. The aggregate simulation disaggregates the unit into the individual vehicles. Later, the unit is reaggregated and the simulation resources consumed by the vehicles become available for other disaggregations.

Summary

Ongoing research to allow aggregate level, time stepped constructive simulations to interoperate with vehicle level, real-time virtual simulations in the distributed interactive simulation environment.

Problems being addressed:

- Movement of units and vehicles between simulation environments.
- Communication of orders, intents, and situation reports between environments.
- Indirect fire combat between entities in the two environments.
- Direct fire combat between entities in the two environments.

Integrated Eagle/BDS-D Phase 2

- Expanding the Interface
  - (1) Add vehicle types: helicopters, aircraft, and ground vehicles.
  - ◦ (2) Track Eagle units within CGF Testbed.
  - ◦ (3) Base Eagle unit detection on BDS-D Line of Sight sightings.
  - ◦ (4) Allow platoon and battalion disaggregation/aggregation.
  - ◦ (5) Include manned simulators.
  - (6) Eliminate fixed Hi-Res area.
  - (7) Make Eagle units visible on Stealth.
  - (8) Control Stealth from CGF OI.
  - (9) DIS compliance.
  - (10) Full call for fire.
- Verification and Validation
  - (11) Use verified parametric Ph and Pk values.
  - (12) Verify the physical models for vehicle dynamics.
  - (13) SME validation of CGF behavior.
- Experimental Research and Development
  - (14) Indirect Fire from CGF entities at Eagle units.
  - (15) Direct fire from CGF entities at Eagle units.
  - (16) Integrate Lockheed Sanders Patriot missile trainer into Eagle/BDS-D system.
  - (17) Expand CGF operator's span of control.
  - (18) Extend Eagle control to BDS-D entities.
  - (19) Partial disaggregation.
  - (20) Add mines.

Intelligent Simulated Forces

N61339-92-C-0045

Intelligent Simulated Forces

Project overview

- Largest single CGF project at IST
- Basic research into CGF behavior generation
- Output of the project is a set of **algorithms for CGF behaviors**
- Algorithms integrated into the IST CGF Testbed, or ?
- IST composed of a set of semi-independent research tasks

ISF research tasks

- **CGF C-to-Ada conversion**
- **Terrain reasoning for reconnaissance planning**
- Efficient line of sight determination
- Unified data structure for terrain reasoning and situational awareness
- Terrain database preprocessing
- Advanced route planning
- Automated mission planning and generation of subordinate unit orders
- Real-time coordination of cooperative behavior
- Intelligent target acquisition, selection, and prioritization
- Machine learning for CGF
- CGF & BR symposia

Terrain Reasoning for Reconnaissance Planning in Polygonal Terrain

Polygonal terrain

Digitized terrain database constructed from polygons.

Base polygons make up the surface of the Earth.
Base polygons are simple, convex, and planar (often triangles or quadrilaterals).
Adjacent base polygons share edges, but are not necessarily coplanar.

Other terrain features are also made of polygons:
1. treelines; sequences of vertical polygons
2. canopies; closed treeline loops, with polygonal tops
3. structures; constructed from polygons
4. roads and rivers; based polygons of different types

Terrain reasoning

Automated analysis of a digitized terrain representation, for the purposes of making behavioral decisions involving the terrain.

Examples:
1. Route planning
2. Seeking cover and concealment
3. Maximizing fields of fire

Reconnaissance planning

Given a terrain area, plan a reconnaissance route of that terrain.
The goals of the route are to:

1. maximize the number of potential defending vehicle positions surveyed
2. minimize the elapsed time before each vehicle position is sighted
3. minimize the total time spent on the route

Important points

Important Points are crucial points in the terrain, such as:
1. ridge crests
2. treeline endpoints
3. structures

Analyzing only the set of Important Points for a piece of terrain is almost as effective, and much easier, than analyzing the entire piece of terrain.

Project summary

1. Studied basic idea of Important Points.
2. Adapted and expanded the Important Points idea for polygonal terrain.
3. Devised the reconnaissance planning task as a test of the Important Points idea.
4. Designed and implemented three new reconnaissance planning algorithms based on the Important Points idea.
5. Designed a experimental test of the algorithms' performance.
6. Conducted the experiment and analyzed its results.

Overview of the All-Points algorithm

1. Given a terrain test area, identify the set of Important Points for that terrain.

2. Select a subset of the Important Point set such that every point in the Important Point set can be seen from at least one point in the subset. Call the subset the "route points".

3. Select a visitation sequence so that the distance travelled in visiting the route points is minimized.

4. Pass the sequenced route points, as a waypoints list, to the IST CGF Testbed's route planner.

Identifying the Important Points

Important Points are placed around terrain features that can block Line of Sight.

Base polygons:	Polygon vertices.
Treelines:	Ends of treelines and midpoints between concavity changes.
Canopies:	Based on the treelines that make up canopy perimeter.
Structures:	Midpoints of each side.

Selecting the Route Points

1. Construct a Line of Sight graph on the Important Points.
   Vertices: Important Points
   Edges: Two Important Points, within 1500 meters, with an unobstructed Line of Sight between them.

2. Use a greedy algorithm to select a subset of the Important Points that can sight the entire set.
   2.1 Choose the Important Point P that can see the most other points (most outging edges), and add it to the route point set.
   2.2 For every vertex Q such that P can see Q, and for every vertex R such that R can see Q, remove the edge (R,Q).
   2.3 Repeat 2.1 and 2.2 until all edges have been removed from the graph.

Sequencing the Route Points

Selecting the order to visit so as to minimize distance travelled is a variation of the Travelling Salesperson problem.

An insertion algorithm is used to find a reasonably good ordering.

1. Sort the route points by descending average distance to the other route points.
2. Start with a fixed starting point.
3. Add each route point, in sorted order, to the sequence by inserting it into the sequence at the point that increases the distance the least.

Inserting the points in the sequence given places the "outliers" first, and then fills in with the clustered points.

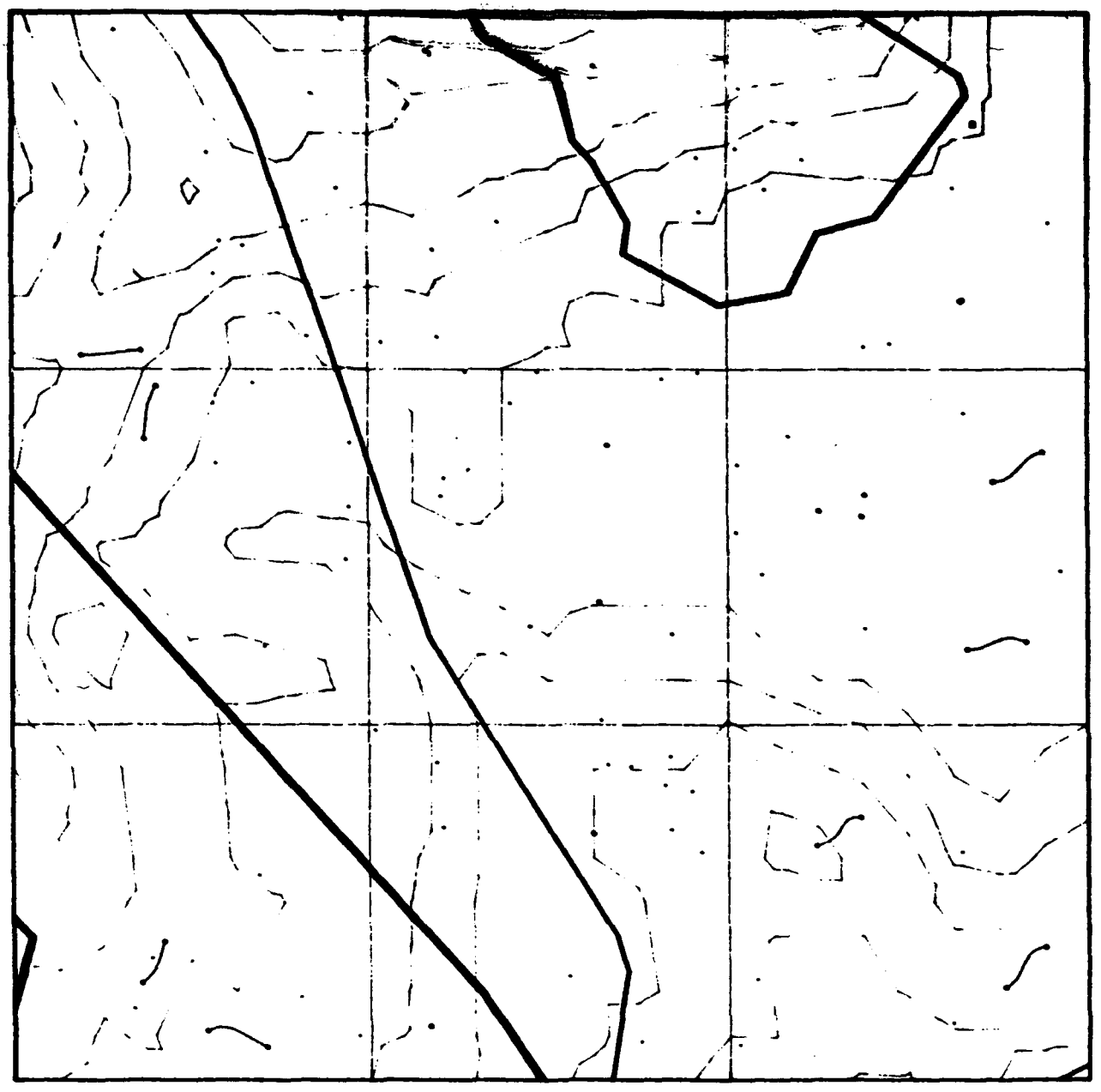
Reconnaissance planning experiment

Terrain test areas:
- Clear, Mixed, Rough

Reconnaissance planners:
- All-Points, Forest Perimeter, Simple algorithms
- SME1, SME2 (military officers)

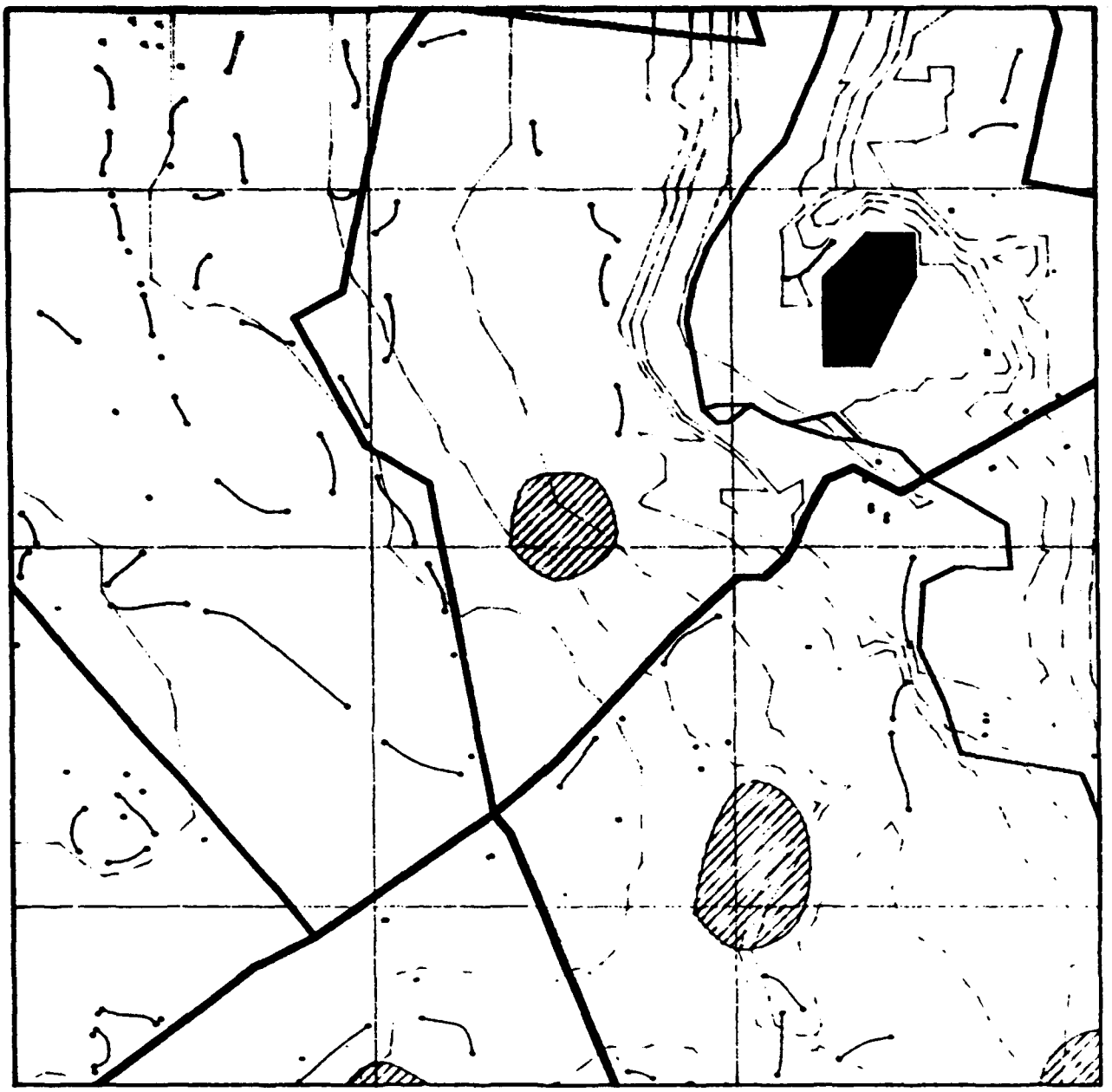
Defensive layouts:
- Defense 1 on Clear, Mixed, Rough (SME3)
- Defense 2 on Clear, Mixed (SME4)
- Defense 2 on Rough (SME5)

5 planners x 3 terrain test areas x 2 defenses = 30 trials



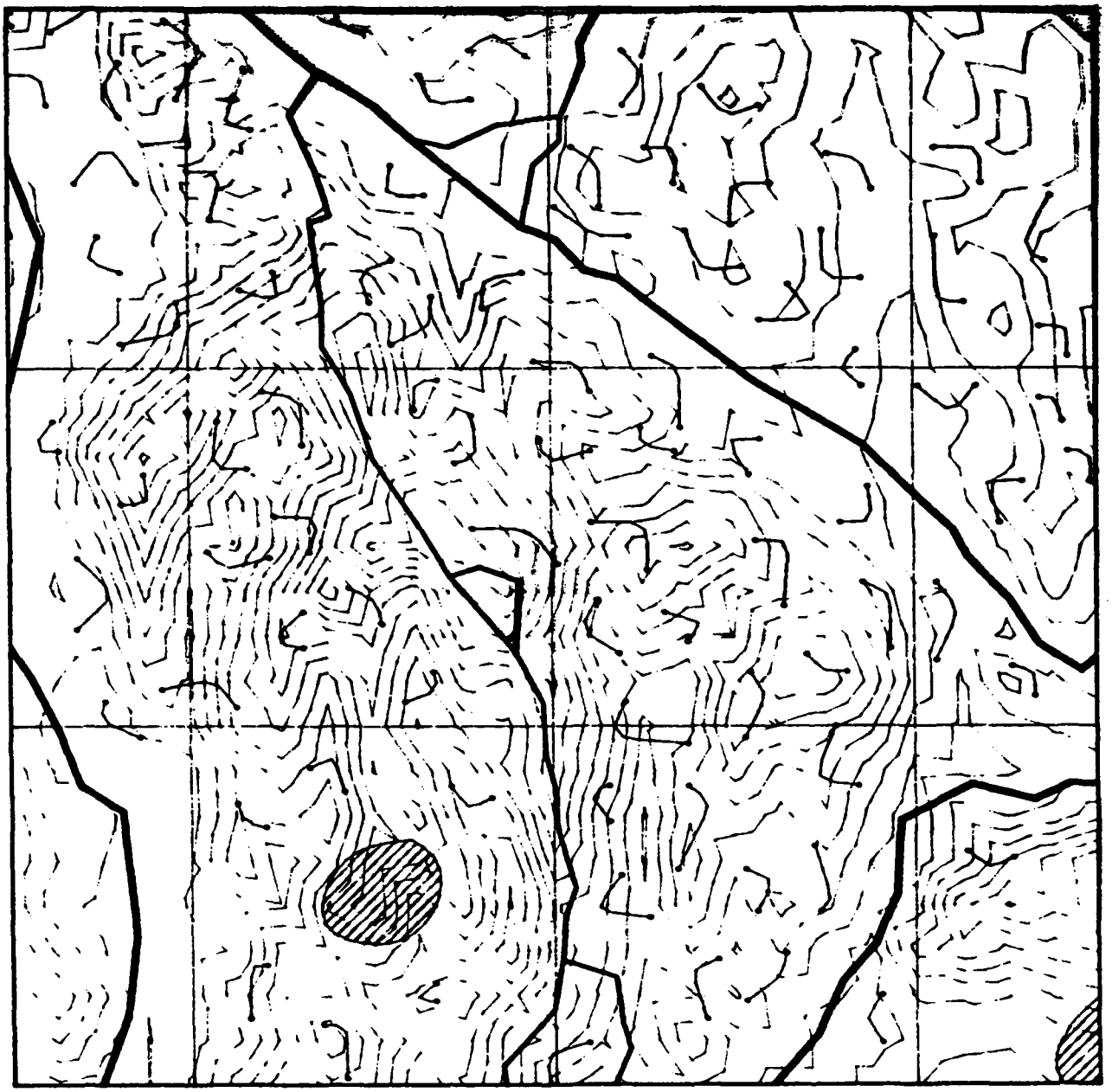
Terrain database:	**SIMNET Ft. Knox**
Location:	**FS055665**
Size:	**3 km x 3 km**
Contour interval:	**5 meters**
Approximate scale:	**1:18000**

Figure 19. Clear terrain test area.



Terrain database:	**SIMNET Ft. Knox**
Location:	**ES845950**
Size:	**3 km x 3 km**
Contour interval:	**10 meters**
Approximate scale:	**1:18000**

Figure 20. Mixed terrain test area.



Terrain database:	SIMNET Ft. Hunter-Liggett
Location:	FQ590905
Size:	3 km x 3 km
Contour interval:	20 meters
Approximate scale:	1:18000

Figure 21. Rough terrain test area.

Subject Matter Experts

5 U.S. Army officers; four from STRICOM, 1 from UCF AROTC.

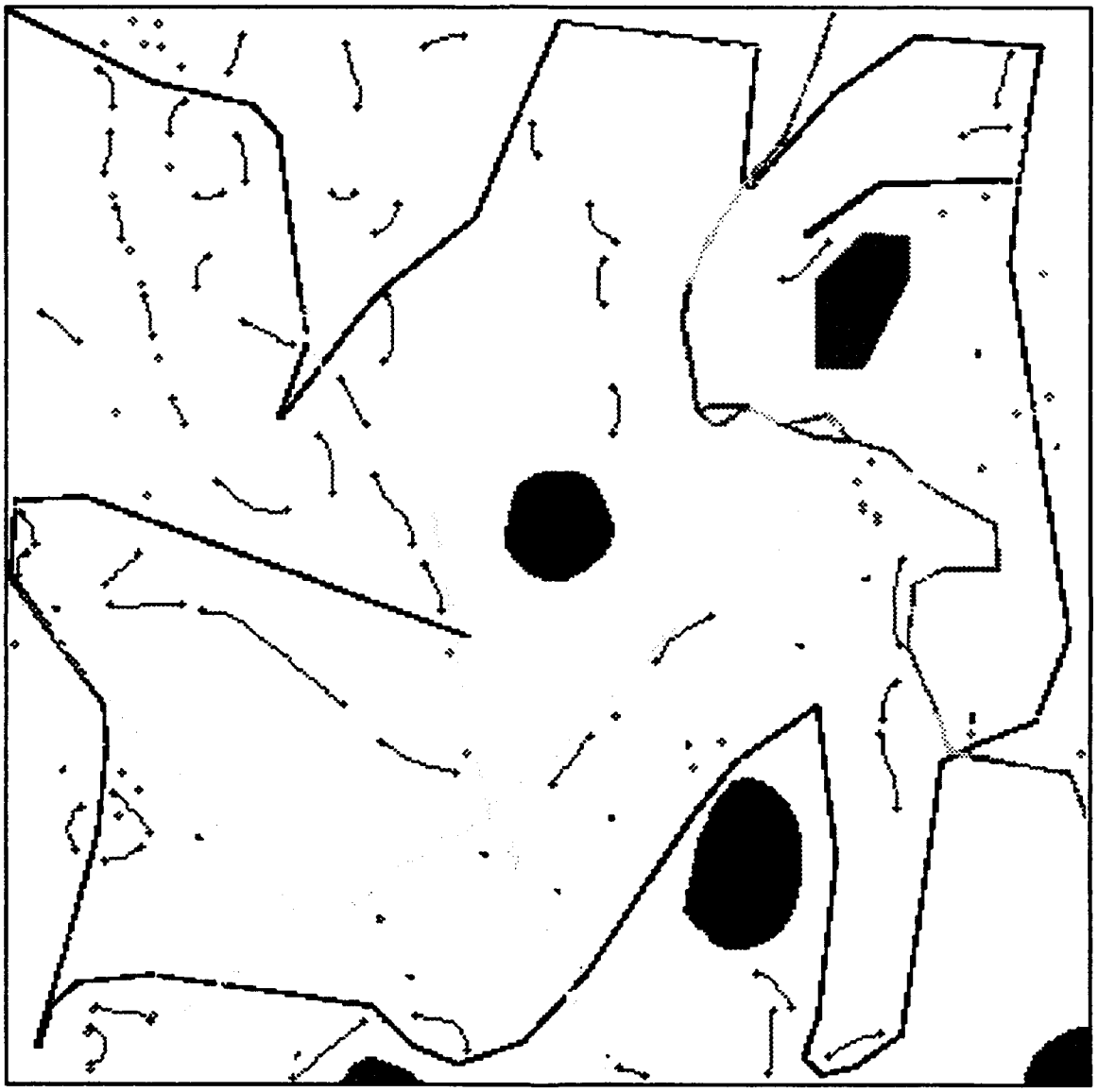
2x CAPT, 2x MAJ, 1x LTC

Extremely well trained and experienced.

- Platoon Leader and Company Commander, 82nd Airborne Division
- Tactics Instructor, USA Infantry School
- S-3, Ranger Training Brigade
- Platoon Leader and XO, 1/73 Armor at National Training Center
- Company Commander, 3rd Armored Cavalry Regiment
- Infantry Platoon Leader and Company XO
- S-2, 2/327 Infantry, 101st Airborne Division
- Commander HHC, 303rd Mechanized Infantry Brigade
- Dual rated Senior Army Aviator
- Company Commander, 268th Attack Helicopter Battalion

The SMEs' expertise was obvious to researchers during experiment.

SME1's route on the Mixed terrain area



All-Points' route on the Mixed terrain area

Defensive layouts

Two defensive layouts for each terrain test area.

Defensive layouts prepared by Subject Matter Experts.

SMEs asked to prepare an OPFOR defense in depth against US attack from the north or west.

Twenty-five target vehicles in each defense.

Experimental data and analysis

Data collected during the trials:
1. Reconnaissance planning times
2. Route completion times
3. Targets sighted
4. Target sighting times

Analysis performed on the collected data:
1. Cumulative time
2. Statistical comparison

Overall performance objective: Sighting more targets earlier is better.

Reconnaissance planning times

Planner	Terrain test area			Total
	Clear	Mixed	Rough	
SME1	2400	1800	3600	7800
SME2	2700	3600	4500	10800
All-Points algorithm	480	2280	2880	5640

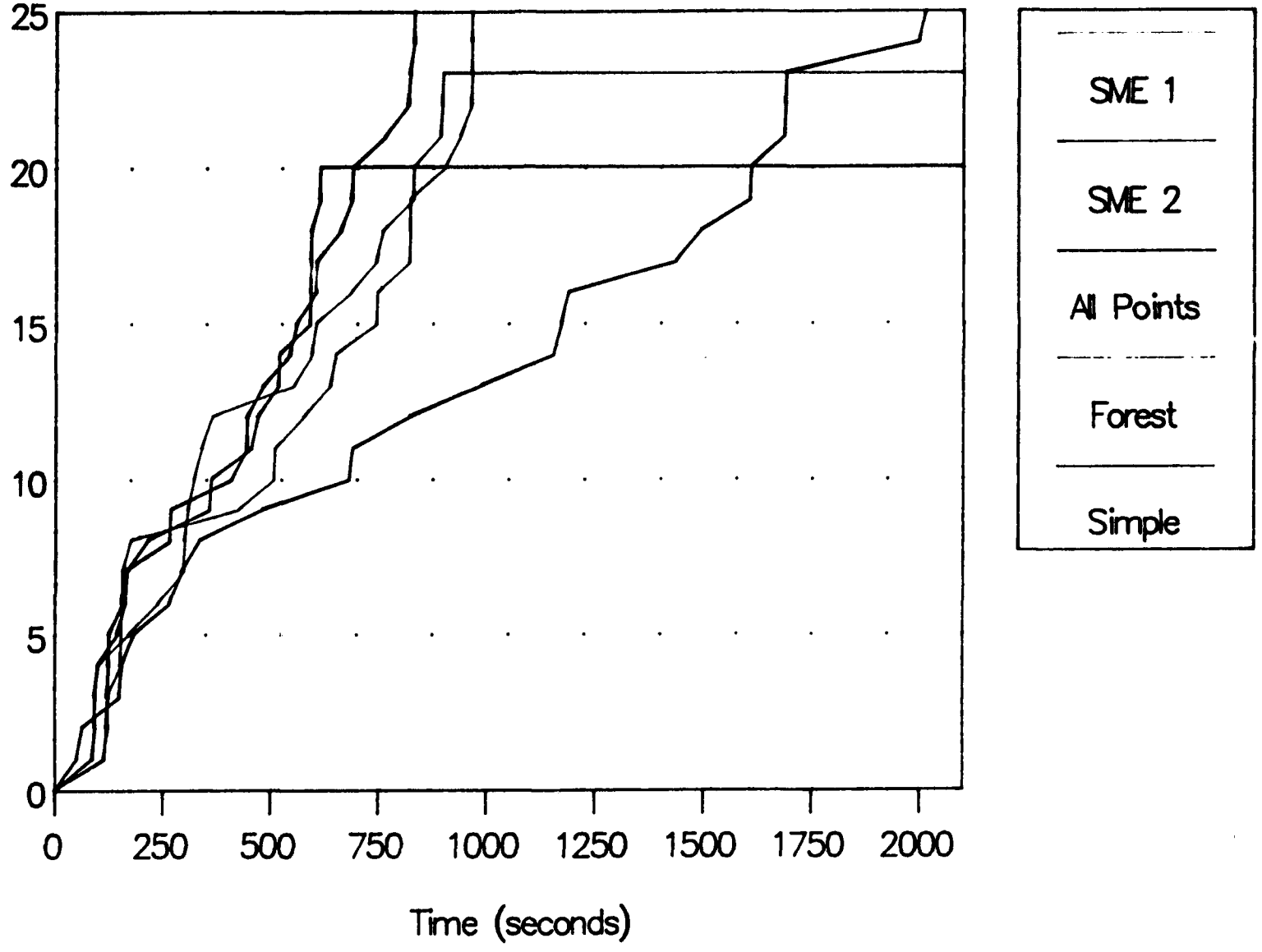
Route completion times

Planner	Terrain test area			Total
	Clear	Mixed	Rough	
SME1	903	1663	1702	4268
SME2	1025	1066	1042	3133
All-Points algorithm	975	1494	2060	4529

Targets sighted

Defense	Terrain test area: Clear 1	Clear 2	Mixed 1	Mixed 2	Rough 1	Rough 2	Total
Planner							
SME1	24	23	25	25	24	23	144
SME2	24	20	25	23	20	24	136
All-Points algorithm	25	25	25	25	25	25	150

CLEAR Test Area: Defensive Layout II



Cumulative time

Base cumulative time:
Sum ot sighting times for each sighted target vehicle and route completion time for each missed target vehicle.

Penalty time:
Additional time added to cumulative time for each missed vehicle.
Based on rule of thumb that missing half the target vehicles is a failed reconnaissance.

Total cumulative time:
Base cumulative time plus penalty time.

	Terrain test area						Total
	Clear		Mixed		Rough		
Defense	1	2	1	2	1	2	
Planner							
SME1	10396	14924	8815	12688	23115	11136	81074
SME2	8454	17664	5942	9717	16134	8526	66437
All-Points algorithm	6560	11645	6869	9702	14991	19262	69029

Statistical comparison

Populations: overall performance of the reconnaissance planners.
Sample data sets: target sighting times observed during the trials.

Wilcoxon Signed-Rank Test (WSRT) statistical hypothesis test
1. Non-parametric; no assumption of normal distribution
2. Internally homogenous datasets, with pairs of treatments
3. Greater resolving power than Sign test

H_0: The two reconnaissance planners being compared performed at a comparable level, in terms of sighting times.
H_1: One of the two planners performed at a substantially better level than the other.

Confidence level: 95%

Statistical comparison (continued)

Terrain	Defense	SME1	SME2
Clear	1	3.10	0.12
	2	2.62	0.58
Mixed	1	-0.29	-2.57
	2	3.51	-2.86
Rough	1	3.54	-3.40
	2	-4.01	-3.81

$Z \leq -1.64$: SME performed better than All-Points
$Z \geq 1.64$: All-Points performed better that SME

The All—Points algorithm performed at a level comparable to or better than the trained military SMEs in seven out of twelve test comparisons, a success rate of 58%.

Summary of results

Reconnaissance planning times:
The All-Points algorithm was faster than the human SMEs.

Route completion times:
The All-Points algorithm planned slightly longer routes than the SMEs.

Targets sighted:
The All-Points algorithm sighted all available targets; both SMEs missed targets.

Target sighting times:
The All-Points algorithm's times were very similar to those of the SMEs.

Summary of results (continued)

Cumulative time:
The All—Points algorithm's value is between the two human SMEs, and is very close to the better SME's value.

Statistical comparison:
In seven out of twelve test comparisons involving the All-Points algorithm, it performed at a level better than or statistically indistinguishable from that of the SMEs.

Conclusion

The All-Points algorithm performed reconnaissance planning at a level comparable with human subject matter experts.

CGF C TO Ada Conversion

The task

Implement a CGF Simulator, written in Ada, functionally equivalent to the IST CGF Testbed Simulator version 6.400.

Goals

Determine whether it is practical to write CGF systems in Ada.

Determine the effects of Ada, relative to C, on software engineering quality in a CGF system.

Determine the effects of Ada, relative to C, on execution speed in a CGF system.

CGF Testbed as of January 1993

CGF has evolved to a reasonably well designed product.

Functions are hidden (static) as far as practical.

No data is exported (data is on the stack or static).

Modules are recognized and binding of all types is minimized.

Since the C mechanism for data hiding is "all or nothing" (being file based) artificial means of enforcing weak bindings are used.

The CGF Simulator consists of approximately 58,000 lines of code (including blanks and comments); it is ANSI C compiled with using a C++ compiler.

January 27, 1993

Ada Conversion Strategy

One extreme approach is to do a C to Ada translation. The other extreme is a complete redesign, needed because Ada offers a whole new approach to software engineering.

Position going into this project:

- A direct translation is of no use.

  The Ada conversion project will produce an Ada product. There is no intent of producing C written in Ada.

- A re-design is not necessary.

  Ada supports the best of the 1980's software engineering techniques. All of the artificial techniques used to enforce cohesion, data hiding, weak binding, etc. can be replaced with direct Ada support.

The C version of the Simulator is playing the role of the Ada product's design document. It is natural to think it would also serve as pseudo-code, and to an extent this is true, but the Ada team is not doing a simple translation, we are building an Ada product.

January 27, 1993

Risks

No project with such great scope is without risk. While an outright failure is not expected, many problems are possible:

- If the Ada team attempts to over-refine the Simulator during the conversion, the project will take too long.

- The assumption that Ada tasks can replace the Simulator executive may not be valid. The Simulator executive was built to fill its role exactly, replacing it with Ada's general tasking support may cause a variety of problems (the most obvious being performance degradation).

- The Simulator uses some facilities which may prove difficult to adapt:

    - High memory support for terrain information (this should be unnecessary with the Ada version).

    - Graphics support.

    - LAN card interfaces.

April 27, 1993

Expectations versus Reality

The quotes are from the slides shown in January.

"The Ada conversion is a natural continuation of the CGF development."

> The conversion is more revolutionary than evolutionary. More fundamental changes are being made than had been anticipated.

"The Ada conversion project will produce an Ada product. There is no intent of producing C written in Ada."

> This is the golden rule for the Ada team. There is nothing that would give away this project's C origins.

"A re-design is not necessary. ... All of the artificial techniques used [in the C version] to enforce cohesion, data hiding, weak binding, etc. can be replaced with direct Ada support."

> The latter statement is true without reservation. The former is more troublesome. Significant parts of the system have been re-designed.

"The use of Ada tasking should make it possible to eliminate the Simulator's built in executive."

> Ada tasking has become the centerpiece of the new simulator.

"The Ada design and run time checks will eliminate the need for many of the facilities hand coded in C...."

> This is largely true.

"If the Ada team attempts to over-refine the Simulator during the conversion, the project will take too long."

> We are making tremendous changes in the system, but our progress is excellent.

Performance Issues

Are Ada executables inherently slow?

Even Ada team members are concerned about the Ada version's performance. Nonetheless, we do not second guess the compiler by choosing an implementation technique based on what we believe will produce tighter code.

Micro versus Macro Efficiency

- The C simulator does low level optimizations which are not practical in Ada.

- Micro-efficiency is less important than macro efficiency.
- The same algorithms are used so we should have roughly the same macro-efficiency and the better Ada organization may increase efficiency.
- Worries about constraint checks (as unnecessary code) are probably overblown.

- The Protocol code may pay a heavy price in efficiency.

- The C version was designed to avoid data copying by contaminating the simulator with some protocol design attributes.
- The Ada simulator makes no compromises and so it will have a much more complex application protocol layer.
- Lower protocol layers may have to do data copies.

- Faster hardware can mitigate efficiency problems.

- Everything about the Ada environment is designed to make it easier for people to design, implement, test, maintain, and enhance software.
- Costs for faster hardware (if necessary) must be balanced against the cost of maintaining and upgrading code written in a more primitive language.

July 12, 1993

Ada Conversion Project Status

Timetable Changes

The Ada conversion team expended all of the time originally estimated for the conversion as of, approximately, July 1, 1993.

- The new estimated completion date for conversion is December 31, 1993.

Development Problems

Protected Mode/Real Mode Interface

Memory Shortage

Library Sizes

July 12, 1993

Code Quality

The Ada team makes a conscious effort to use the Ada facilities most likely to lead to the highest code quality. We continue to improve code quality wherever practical. Examples:

- In spite of our efforts, we had used inappropriately broad types for some quantities. There was only one speed type, which was used for both DI and missiles (!). We now have types appropriate for the individual entities, eliminating the Mach 1 DI.

- In looking into the last item, we recognized a whole class of data and types which were over-generalized. As part of changing the speed-type, other overly general structures were eliminated.

- The Ada-CGF code uses generics wherever appropriate to share code. In spite of the break up of encompassing data structures, the behavior and dynamics code remains flexible because it is instantiated appropriately for the entity in question.

It may be worth noting that changes, such as noted in these examples, have often lead to the subjective, but comforting, "ah-ha" response.

Consider dynamics:

- Parts of dynamics code unique to a vehicle is written in the package for the vehicle.

- Parts to be shared but which are dependent on the vehicle's associated data types is generic.

- The remaining code consists of procedures which interact with vehicle instances through parameters. The code does not need to accept overly general data structures, nor does it have to ask what is being processed ("am I a DI?") by examining its own data.

July 12, 1993

Lessons Learned

Using Ada on Personal Computers

Insure your PC is up to the task.

We expected to do all the development on 386 33 Mhz. PCs with less than 4 megabytes of memory. This would have been impossible; by adding memory the task would not be impossible, but would have been impractical. Whether a task is practical on a given machine depends, of course, on the task; For our task 486 machines running at 66 Mhz with 4 megabytes are inadequate; with additional memory we should be able to complete the project.

July 12, 1993

Using Ada for CGF

The Ada CGF Software is of higher quality then the C rendition

- The language encourages good software practices.
- The enforcement of software discipline is much more powerful than most people realize.
- It is difficult to imagine anyone seriously disputing the Ada team's claim that the Ada code is of much higher quality than the C version.

Ada Facilities are well suited for simulations

- Ada tasking eliminates the need to build custom executives.

  At least one member of the Ada team believes the tasking capability alone is sufficient reason to make Ada the language of choice for CGF.

- Generics are powerful tools for customizing tasks for various entities.

  A significant amount of code can be shared by DIs and tanks in spite of their obvious differences. Thanks to generics, the code takes on the appropriate characteristics for the entity.

- Ada support code (such as constraint checks) finds errors that would otherwise be missed.

  The Ada team has found places in the C code where uninitialized memory is used for floating point values. In most cases the nature of the arithmetic done with this memory allows the simulation to continue, but this is a time bomb. Ada trapped the error immediately.

October 13, 1993

Intelligent Simulated Forces
CGF Ada Conversion Experiment Status

Over 58,000 lines of tested Ada code have been written.

This is up from 35,000 lines at the last quarterly review. To completely represent the 6.4 C version will probably take about 70,000 lines of code.

External data and events (files and keyboard) are correctly used in almost all cases.

Most of the protocol code is complete.

About a month ago our consultant was able to solve the protected/real mode problem. After some rapid perturbations a usable, though limited, version of a packet driver interface was delivered, and has been in use ever since.

Our consultant has recently developed a more robust and complete version of the driver support which has yet to be integrated.

Most of the higher layer protocol support is in place.

Vehicle dynamics are complete.

The bulk of the behavior code has been developed.

October 13, 1993

The FSM Architecture

The FSM design is fundamental to intelligent behavior, and yet the FSM implementation is radically different in the Ada and C versions of the Testbed. Many key questions raised in discussing conversions to Ada were addressed when designing the FSM support.

Here are some of the key features required by the FSM implementation and how they are handled by the two systems

FSMs are tasks

FSMs must be able to receive messages and act on them. For example, an FSM may need to be awakened when another FSM completes or when a timer expires.

C: All tasking is done through a custom executive. "Ordinary" (non-FSM) tasks have control blocks (for their data). FSMs, and their associated data, are accessed through the control block of their parent. Messages for FSMs are delivered by locating the parent task and then locating the FSM.

Ada: FSMs are subtasks of the entity which invoked them. Messages for FSMs are delivered to the parent which then delivers the message to the FSM.

In many ways there is a good parallel between the two versions in this mechanism. The key difference is that the Ada version uses the Ada features to maintain the tasks.

FSMs modify the data of their parent task

For example, when a vehicle uses an FSM to turn, the FSM must periodically adjust the vehicle's yaw so the vehicle will turn gradually.

C: The FSM is given a pointer to the parent task's control block. This is practical since tasks in the C system all have control blocks. Because the FSM can directly access the vehicle's control block it has access to all of the vehicle's data.

Ada: There are no control blocks in the Ada implementation. The vehicles (ACTORs) are tasks which have ordinary local data. FSMs are generic and are instantiated with OUT parameters for data which the FSM must modify. The FSM is given access only to data which it needs.

FSMs retain state information from event to event

State machines run through a sequence of states; events are interpreted in the context of the FSM's active state.

C: An FSM's state is represented by a function. When an FSM is activated to process an event its current state is called with an indication of the event as well as a data structure which serves as a control block for the FSM (its data area).

Ada: FSMs are implemented as tasks. Since they are tasks they retain their local data without special mechanisms. A local enumeration is used to track the current state. A switch on the current-state variable yields the proper context.

FSMs must be able to report results to other entities

If a vehicle wishes to fire a weapon, the FSMs used to do this will ultimately activate an FSM to aim the weapon before it is fired. When the aiming is complete the parent needs to be informed so firing can commence.

C: a general purpose FSM reply mechanism is implemented. The list of possible replies is set out as an enumerated type used by all FSMs. Along with "SUCCESS," replies exist to cover various anticipated scenarios (e.g., "IMPOSSIBLE" and "EXTENDED_ROUTE_NEEDED").

Ada: A general reply mechanism is defined. The FSMs, as ordinary tasks, define replies just as ordinary ACTORs do. The possible replies are defined by the task sending the reply (in its specification). This way, only replies that make sense for the task under consideration are defined, and as much data as makes sense can be associated with the reply. Hypothetically, AIM could tell its parent it is aimed and the direction of the target or, in case of failure, exactly why it failed (perhaps the target was no longer visible).

FSM re-use may require associated FSMs to be terminated

If a vehicle is told to route to a point it will invoke a series of FSMs to carry out this behavior. These FSMs may invoke an FSM to cause it to face the appropriate direction. Should some other behavior request the vehicle to face another direction, there is a clash between the requests. This is resolved by giving the new request priority, which forces the route to be abandoned. In so doing, the various routing FSMs are all shut down.

C: a series of functions is used to recursively kill FSMs by removing them from the list of active FSMs for an entity. The mechanics of this implementation prevents 2 copies of an FSM from running at the same time (a vehicle with two weapons cannot aim them at the same time if they share any FSM to accomplish aiming; vehicles cannot launch two missiles at the same time because missiles rely on FSMs).

Ada: Entities which use FSMs have a subordinate task which tracks which FSMs are active and who started them. Associated data determines how many copies of an FSM can run at once. Starting new FSMs will, when appropriate, bring down the necessary FSM subtree.

Recovery of dynamically allocated memory has made FSM shutdown complex in the Ada version which, in turn, has complicated the FSM hierarchy control.

October 13, 1993

Miscellaneous

Compilation Time

System compilation time is about 2 hours for a full build (58,000 lines, 229 files). Large disk caches are used (on the order of 16 Megabytes) during compilation; this is essential to keep compilation time down.

Library Sizes

Our system is configured across 4 libraries, ranging in size from 21 Megabytes (the main development library) down to 1/2 Megabyte (for the protocol library); the total size is about 27 Megabytes.

October 13, 1993

Conclusions

The Ada CGF Testbed has developed to the point that some preliminary conclusions can be made. These conclusions have not been quantified but are opinions.

CGF systems can be written in Ada

The Ada CGF Testbed is already sufficient to demonstrate this.

Ada's software engineering features can be used to produce higher quality code.

- strong type checks not only catch errors, they encourage proper typing (our tanks are not capable of speeds as great as a missile's) and can simplify code.

- Packages encourage strong encapsulation and localization. By being careful the Ada team has kept the complexity of package specifications to a minimum, avoiding unnecessary binding. Ada makes people think about these issues.

- Generics encourage code re-use and, simultaneously, customization.

Future production CGF systems should be done in Ada

Production products should be given the benefit of Ada's maintainability.

Converting an existing system from C to Ada should be approached as a redesign, not a "translation"

Without this approach the benefits of Ada will not be realized. Attempts to preserve aspects of the C CGF Testbed proved counterproductive.

Should Ada be used for research?

There is a big difference between exploratory programming and production programming. Most people we have asked will agree that Ada is the right choice for production programs. Whether Ada is the right choice for research has yet to be answered.

October 25, 1993

Intelligent Simulated Forces
CGF Ada Conversion Experiment

Preliminary Performance Evaluation
October 25, 1993

Part of the Ada team's resources were redirected to carry out a performance evaluation of the existing Ada CGF Testbed. This was done on October 20, 21, and 22.

The Evaluation Parameters

Three dimensions of the CGF Simulator's performance were considered for evaluation.

- The number of vehicles that can operate locally (i.e. on a single copy of the Simulator).
- The number of remote vehicles which can be serviced.
- Robustness under internal computational load.

The third item refers to observation of the Simulator performance as the number of internal computations is varied. This was accomplished by varying the frequency of Line Of Sight (LOS) computations, an expensive process.

Recognition of Simulator Stress

A Simulator under stress may fail in a number of ways, including:

- A breakdown in behavior.
- A system crash.
- The loss of incoming traffic (dropped packets).

Custom Testing Tools

To gain maximum control over the variables in the tests, and to simplify the effort, a program was built which transmits a valid appearance packet at a user selected rate (up to 425 packets a second).

Increasing the rate of transmission effectively loads the system under test with network traffic, but this is not the same as increasing the number of vehicles because the packets all represent the same vehicle at the same location.

For the purposes of discussion, each 5 packets per second is said to represent a "pseudo-vehicle." Idle remote vehicles produce a packet every 5 seconds, moving vehicles produce less than 7 packets a second.

Preliminary Test Adjustments

All tests were run with 12 vehicles:

- This decision came about for several reasons, the key being that the rated capacity of the C Simulator is 12. This decision also simplified the testing and the presentation of results.

The error threshold must have at least two dimensions:

- It was discovered that the Ada Simulator dropped packets even under mild stress and so our tolerance threshold had to have a "dropped packet" dimension.

- The C Simulator was never seen to drop packets, so a visual threshold was established. If a routing vehicle missed points the Simulator was viewed as past the tolerance threshold. The Ada Simulator also shows this form of stress.

Test Summary

A script was run which generated 11 turning vehicles and another vehicle following a reasonable complex route. A LOS rate was established and the packet load was varied to locate the threshold for each Simulator. The results are summarized in the following tables and then graphically.

Evaluation Results for the C Simulator

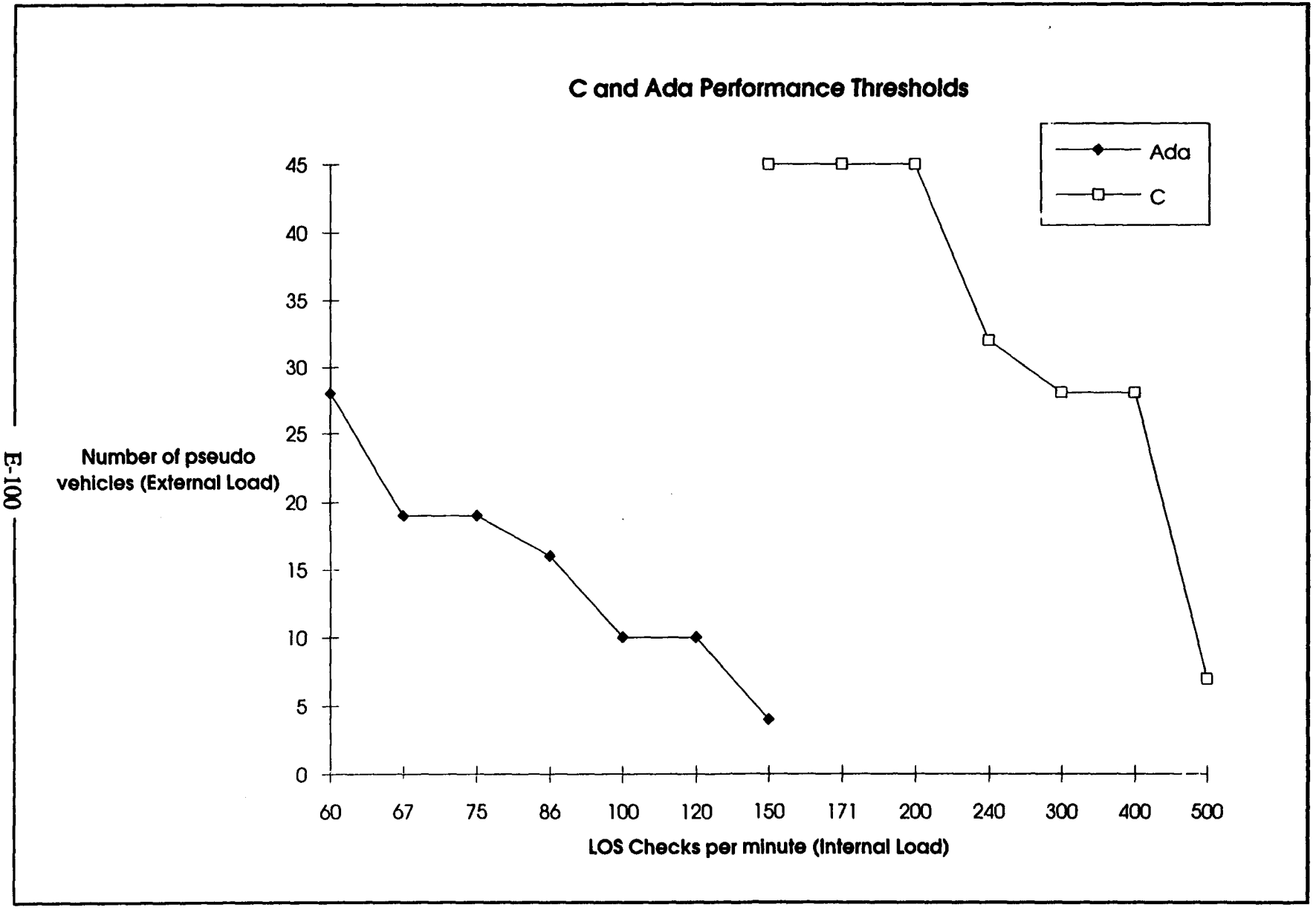
	LOS	packets/sec (ps-vehicles)	% discarded	visual degradation
	120	225 (45)	0 %	no
•	150	225 (45)	0 %	yes
	150	160 (32)	0 %	no
•	171	225 (45)	0 %	yes
	171	160 (32)	0 %	no
•	200	225 (45)	0 %	yes
	200	160 (32)	0 %	no
	240	225 (45)	0 %	yes
•	240	160 (32)	0 %	yes
	240	131 (28)	0 %	no
	300	160 (32)	0 %	yes
•	300	131 (28)	0 %	yes
	300	97 (19)	0 %	no
	400	160(32)	0 %	yes
•	400	131(28)	0 %	yes
	400	97(19)	0 %	no
	500	36 (7)	5 %	yes
•	500	38 (7)	5 %	yes
	500	40 (8)	5 %	no

LOS: Number of LOS computations done per minute.
packets/ sec: Number of packets received per second.
The paranthetical number indicates the number of pseudo-vehicles.
Visual Deg.: Was visual degradation noticed?

Evaluation Results for the Ada Simulator

	LOS	packets/sec (ps-vehicles)	% discarded	visual degradation
	60	160 (32)	> 50 %	no
•	60	131 (28)	~ 10 %	no
	60	97 (19)	< 10 %	no
	67	131 (28)	> 10 %	no
•	67	97 (19)	~ 10 %	no
	67	82 (16)	< 10 %	no
	75	131 (28)	> 10 %	no
•	75	97 (19)	~ 10 %	no
	75	82 (16)	< 10 %	no
	86	97 (19)	> 10 %	no
•	86	82 (16)	~ 10 %	no
	86	74 (15)	< 10 %	no
	100	66 (13)	> 10 %	no
•	100	51 (10)	~ 10 %	no
	100	46 (9)	< 10 %	no
	120	66 (13)	> 10 %	no
•	120	51 (10)	~ 10 %	no
	120	46(9)	< 10 %	no
	150	24(5)	> 10 %	no
•	150	22(4)	~ 10 %	no
	150	20 (4)	< 10 %	no

LOS: Number of LOS computations done per minute.
packets/ sec: Number of packets received per second.
The paranthetical number indicates the number of pseudo-vehicles.
Visual Deg.: Was visual degradation noticed?


C and Ada Performance Thresholds
Ada
C
45
40
35
30
25
20
15
10
5
0
Number of pseudo vehicles (External Load)
60
67
75
86
100
120
150
171
200
240
300
400
500
LOS Checks per minute (Internal Load)

October 25, 1993

Conclusions

The Ada version is slower.

Possible Reasons for the slower execution include:

- The Ada version is a new product. The C version is a mature product and has undergone continuous improvement often aimed at enhanced efficiency.

- The Ada packet support is in the form of a TSR standing between the Ada Simulator and a packet driver. This code is very new and may contain serious problems. Architectural problems on the PC may be causing us problems.

This cannot be the complete problem, however, since the Ada Simulator can be stressed without any traffic (and as few as 4 vehicles).

- The Ada team has made no compromises for efficiency. All checking is on (except in one checksum routine where integers are intended to overflow), tasks are used to protect critical sections, and there are no mixed language sections (the TSR, which is written in C, is not bound with the Ada program; it is simply a gateway to the packet driver).

System profiles may uncover some expensive areas.

- The Alsys compiler, and PC-Ada compilers in general, are not as mature as C compilers.

Can the Ada Simulator be improved?

Of course. It is an open question as to how much it can be improved. If it turns out the TSR is causing problems, we can hope for major gains without compromising the design.

It would be a mistake to optimize the Ada Simulator by compromising the software engineering quality of the product.

APPENDIX F

AUTOMATED FORCES TECHNOLOGY PROGRAM BRIEFING (ADVANCED RESEARCH PROJECTS AGENCY)

Advanced Research Projects Agency

AFOR

Automated Forces Technology Program

Program Overview and Development Plans

October 1993

Advanced Research Projects Agency

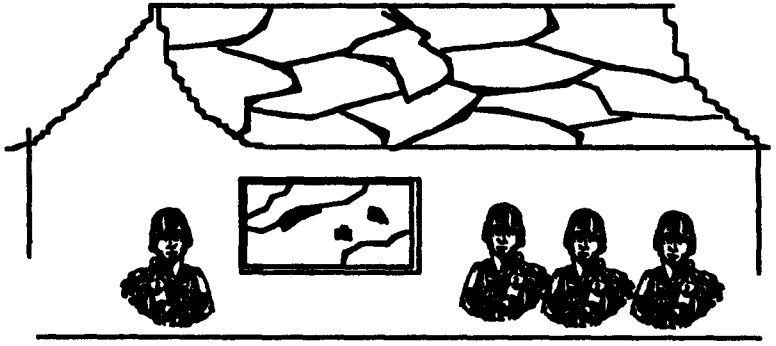
AFOR

Automated Forces Technology Program

- Vision
- Program Plan Overview
- Higher Echelon Command AFOR
  - Technical Approach
  - Program Plan

Vision

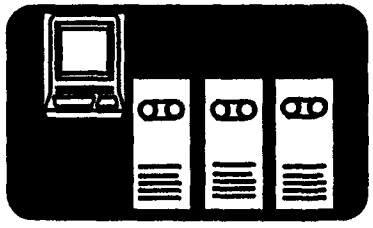
AFOR



Appear the same to the commander and staff

Real wartime action

Simulated Entities

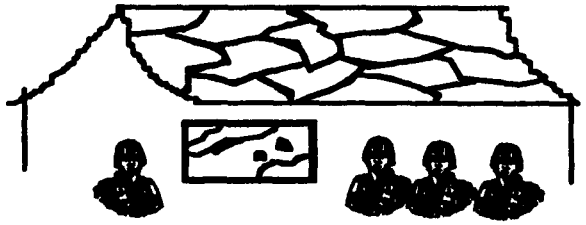


- Represent a Joint Task Force operation in virtual simulation
- Commanders and their staffs operate in exercises just as they would in wartime, except that
  - Subordinates and enemy forces will be represented in software as
  - Entity based virtual forces
    - » Battle effects will be resolved at the level of individual weapons systems
    - » Multi-echelon command, control and communications will be explicitly represented in virtual simulation

Advanced Research Projects Agency

Automated Forces Technology Challenge

AFOR



Technology Gap

Virtual Simulation Technology Today

- **Small Unit Commanders**
- **Selected Vehicles**

- Technology exists to represent selected, semi-automated platform level entities and small unit commanders in software
- The technology gap:
  - levels of command between small units (platoon) and higher echelon commanders
  - capability, diversity, and autonomy of small units and platform entities
- This program will address the technology to fill this gap

Goal of the Automated Forces Program

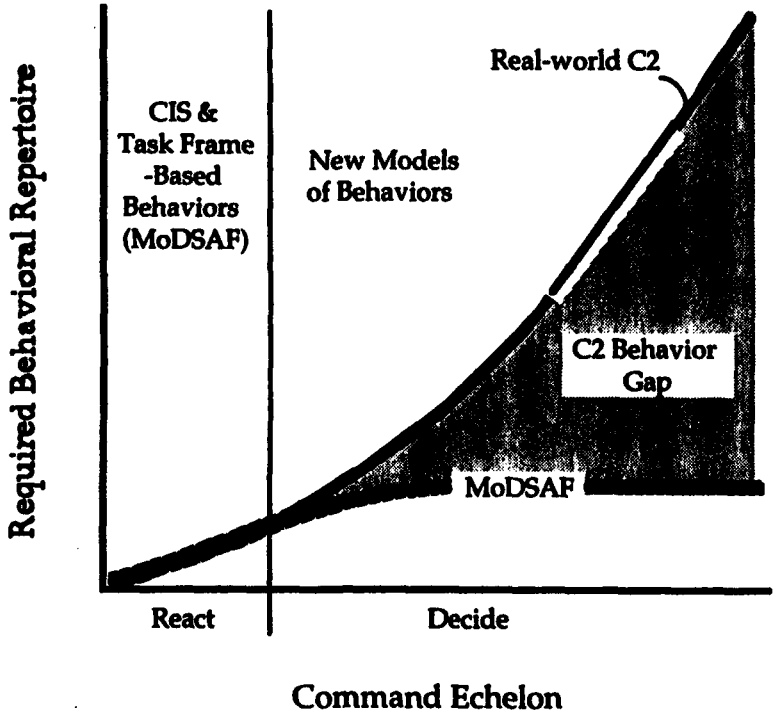
- **Develop the technology to represent higher echelon command, control and communications (C3) in entity-based virtual simulation**
- **Increase the capability and diversity of virtual platform-level entities needed to represent combat of today and tomorrow**

Program Plan Overview

The AFOR program has four technology panels which address key technology challenges in applying virtual simulation to each of the services:

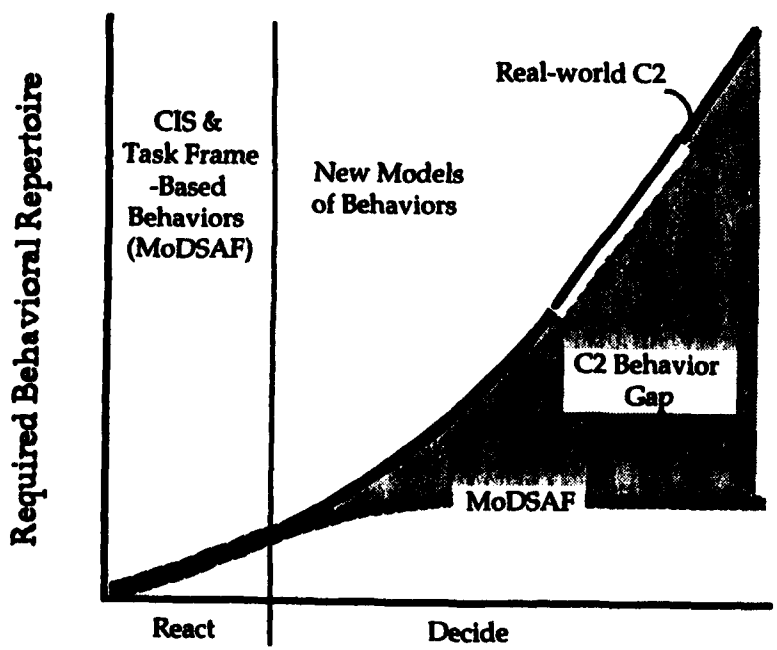
- **Higher Echelon Command and Control (Army)**
- **Realistic Representation of Dismounted Infantry (Marine Corps)**
- **Integration of AFOR with Real World C3 Systems (Air Force)**
- **Representation of systems of systems and EW in DIS (Navy)**

Dimensions of Higher Echelon Command AFOR



- **Requires decision-making behavior**
  - **Differs in kind from reactive entity and small unit commander SAF behavior**
  - **Complex process**
    - » **System of multiple interrelated decision-makers (e.g., hierarchy of command)**
    - » **Information rich; requires ability to evaluate information (e.g., identifying tactically relevant information)**
- **Broad in scope**
  - **Need to focus on key components if problem is to be tractable**
- **Requires a general, extensible approach**

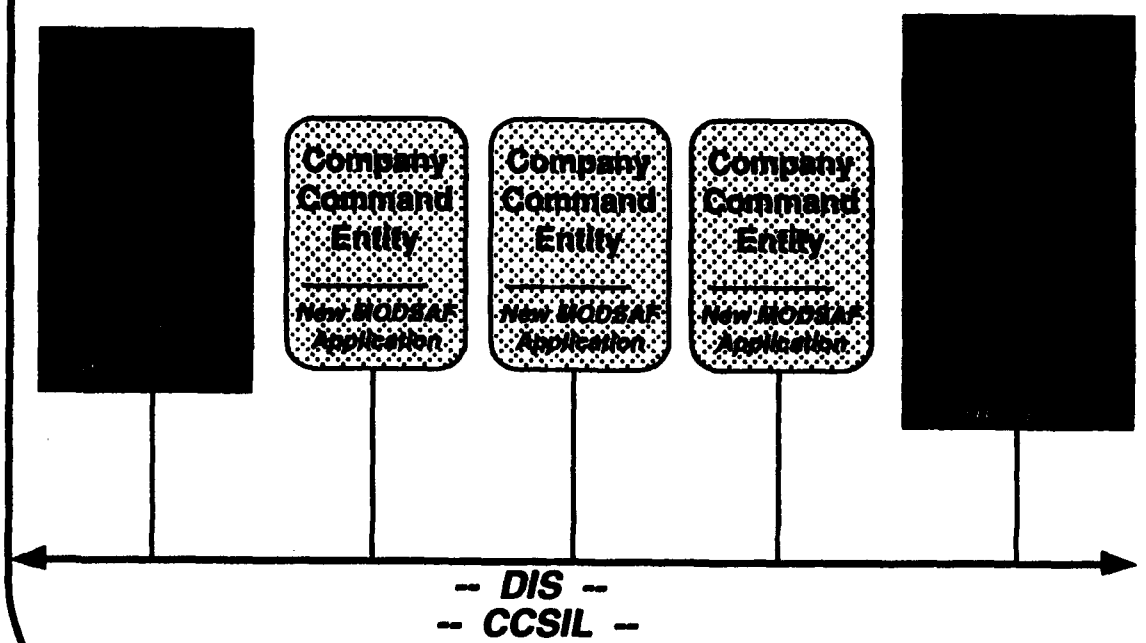
AFOR


Required Behavioral Repertoire
CIS & Task Frame -Based Behaviors (MoDSAF)
New Models of Behaviors
Real-world C2
C2 Behavior Gap
MoDSAF
React
Decide
Command Echelon

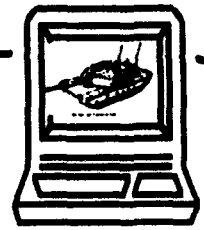
General Technical Approach

AFOR

Extend the SIMNET/DIS paradigm to incorporate the C3 process

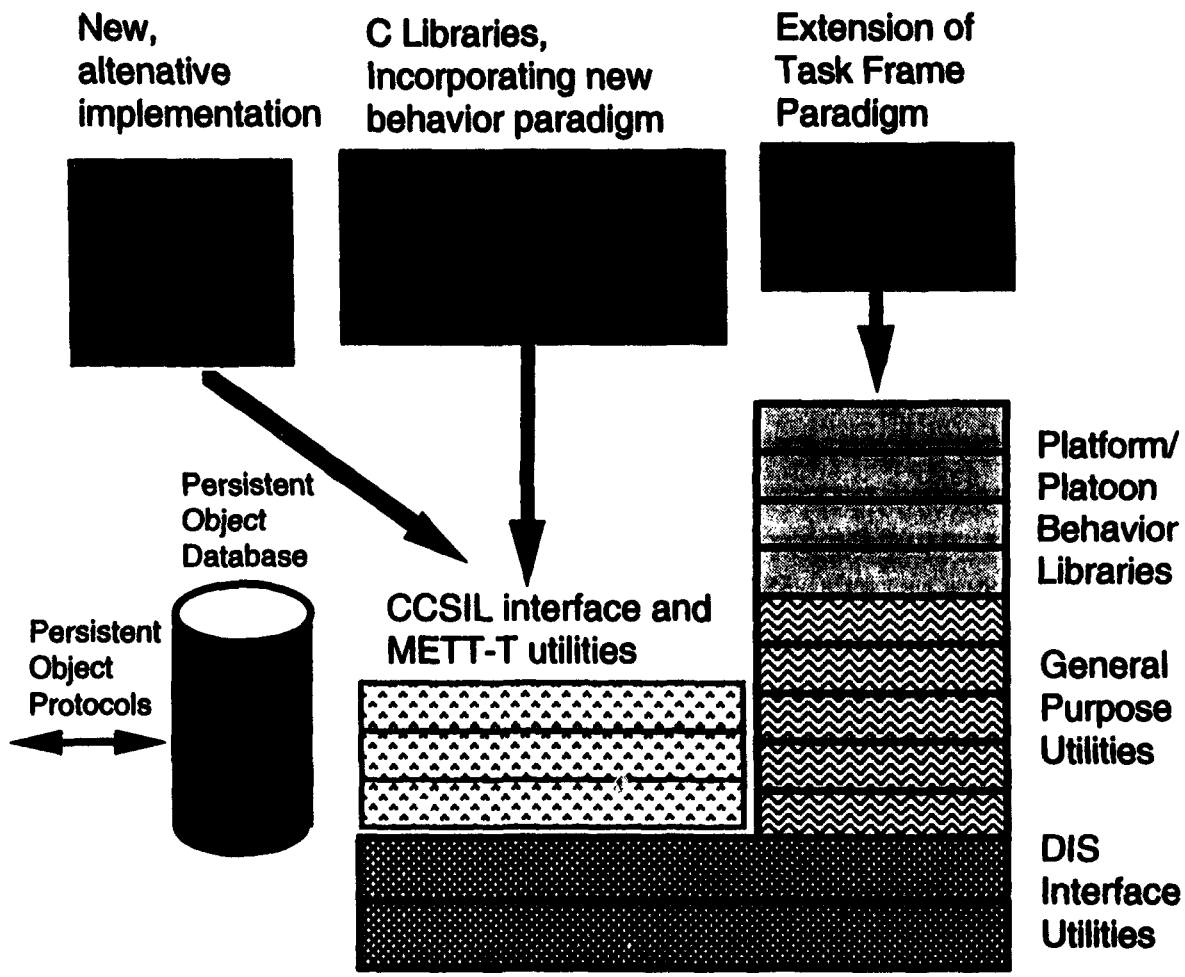


- **Close match between the real world environment of C3 and the SIMNET/DIS paradigm**
  - **Command entities**
  - **Command and Control Simulation Interface Language (CCSIL, pronounced "cecil")**
  - **Command and control decision-making behavior within command entities**
  - **C3 information flow among entities**



AFOR

Development of Higher Echelons Command Entity Forces



Extension of SIMNET/DIS Paradigm

- **Command Entities**
  - **Commanders can be represented as DIS entities**
  - **Command entities represent virtual decision-makers at multiple echelons**
  - **Command entities have physical features as battlefield entities**
    - » **Subject to battlefield effects**
- **Command and Control Simulation Interface Language (CCSIL or "Cecil")**
  - **Interactions among command entities can be represented as message exchanges**
  - **CCSIL will parallel the real-world information exchanged among command decision makers**
  - **Includes**
    - » **Routine, scheduled information (operations orders, situation report) and situation triggered exchanges (spot reports and frag orders)**
    - » **Objective state changes ('unit strengths') and subjective, commander assessments**

Continued...

Advanced Research Projects Agency

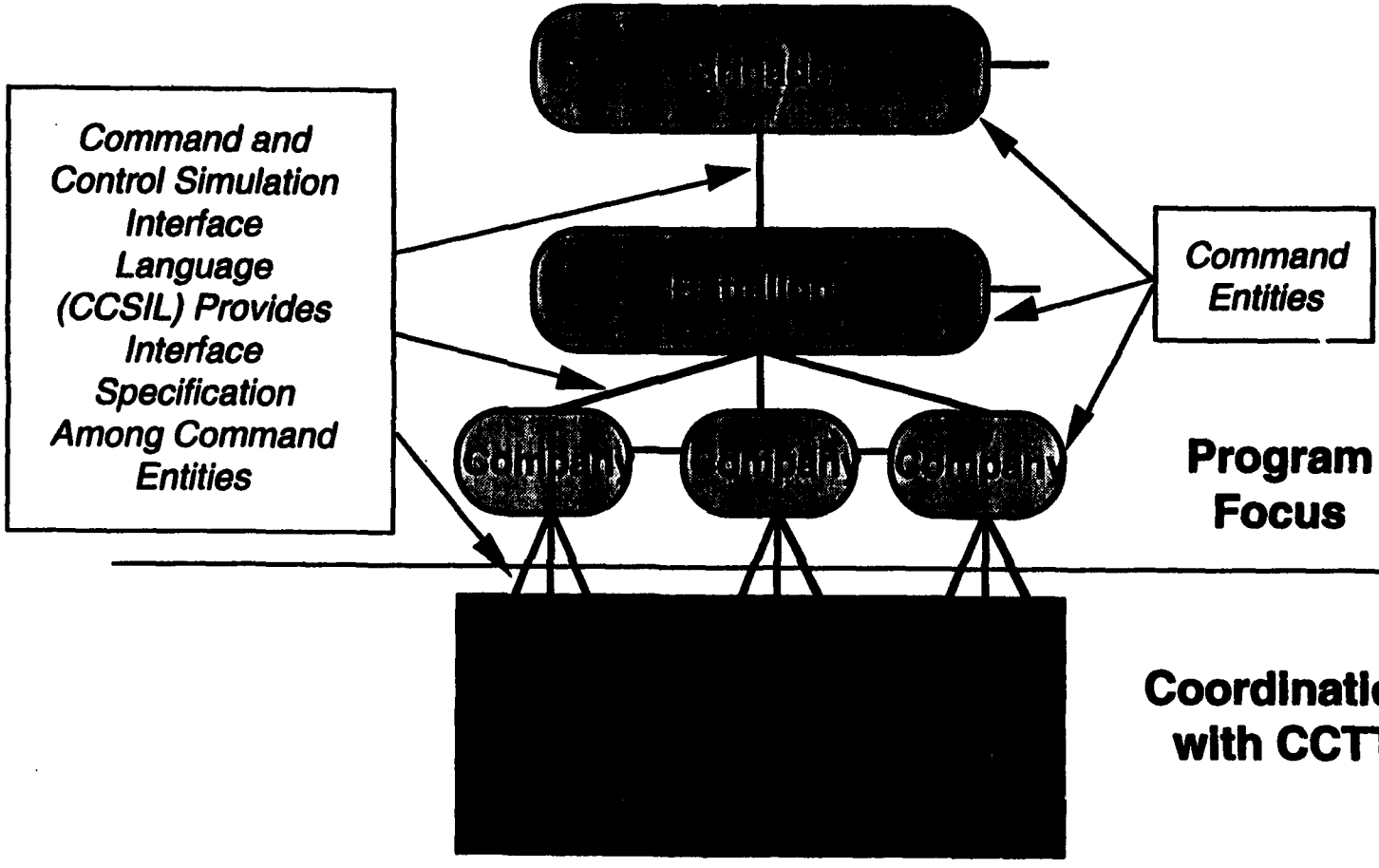
Extension of SIMNET/DIS Paradigm

- **Command and Control Decision-Making Behavior**
  - **Command entities represent command and control decision-making behavior**
  - **Command entities operate within the constraints of CCSIL *information flows***
  - **Selective fidelity of command behavior makes problem tractable**
  - **Different command entities *may represent behavior using different technical approaches***
- **C3 Information Flows**
  - **CCSIL information exchanges will parallel real world communications**
    - » **Procedurally**
    - » **Subject to battlefield effects**
  - **Approach will extend DIS broadcast and throw away paradigm**

Technical Concept Illustrated

AFOR

Command and Control Simulation Interface Language (CCSIL) Provides Interface Specification Among Command Entities



Command Entities

Program Focus

Coordination with CCTT

Overview of Development Plan

- **Level 0 Concept Development and Systems Integration**
  - **Objective: Develop technical and conceptual base for user prototyping**
  - **Includes:**
    - » **Proof-of-principle (POP) prototyping of CCSIL with ground application**
    - » **Integration environment for user proof of concept prototypes**
    - » **Concept/Technology review**
- **Level 1 Proof-of-Concept (POC) Prototyping**
  - **Objective: Develop user prototypes to demonstrate viability of technology**
  - **Includes on-site experimentation with ground, command entity prototypes**

Continued...

Overview of Development Plan

- **Entity and Infrastructure Development**
  - **Objective: Develop the infrastructure necessary to support C3 AFOR technology development**
  - **Includes:**
    - » **Small unit commander and vehicle automated forces**
    - » **Terrain databases**
    - » **User testbed facilities for knowledge acquisition and experimentation**

Level 0 Concept Development and Integration: FY94

AFOR

- **Define C2 model with focus on ground operations**
  - Develop a strawman CCSIL for platoon to battalion exchanges
  - Develop functional specification for ground maneuver company command entities
  - Evaluate current and planned vehicle and platoon level SAF (e.g., MoDSAF 1.0 application, CCTT)
  - Identify critical technical issues
  - Develop functional specification for ground maneuver battalion command entities
- Conduct Concept/Technology Review
- Construct implementation concept based on
  - Assess MoDSAF as a software environment for development of command entity AFOR
  - Assess options for implementing CCSIL exchanges (e.g., DIS, Persistent Object Protocol)
  - Examine interface to vehicle and platoon level SAF

Continued...

Level 0 Concept Development and Integration: FY94

- **Proof-of-Principle (POP) Laboratory Demonstration**
  - **Components**
    - » **Adapt BN Automated TOC as BN AFOR workstation**
    - » **Adapt LADS/SAIC low echelon SAF to use CCSIL**
    - » **Develop skeletal company command entity**
    - » **Verify technical approach prior to POC development**
  - **Provide environment for POC development integration and test**

Advanced Research Projects Agency

Level 0 FY94 Tasks and Schedule

AFOR

Task I: C2 Model Definition

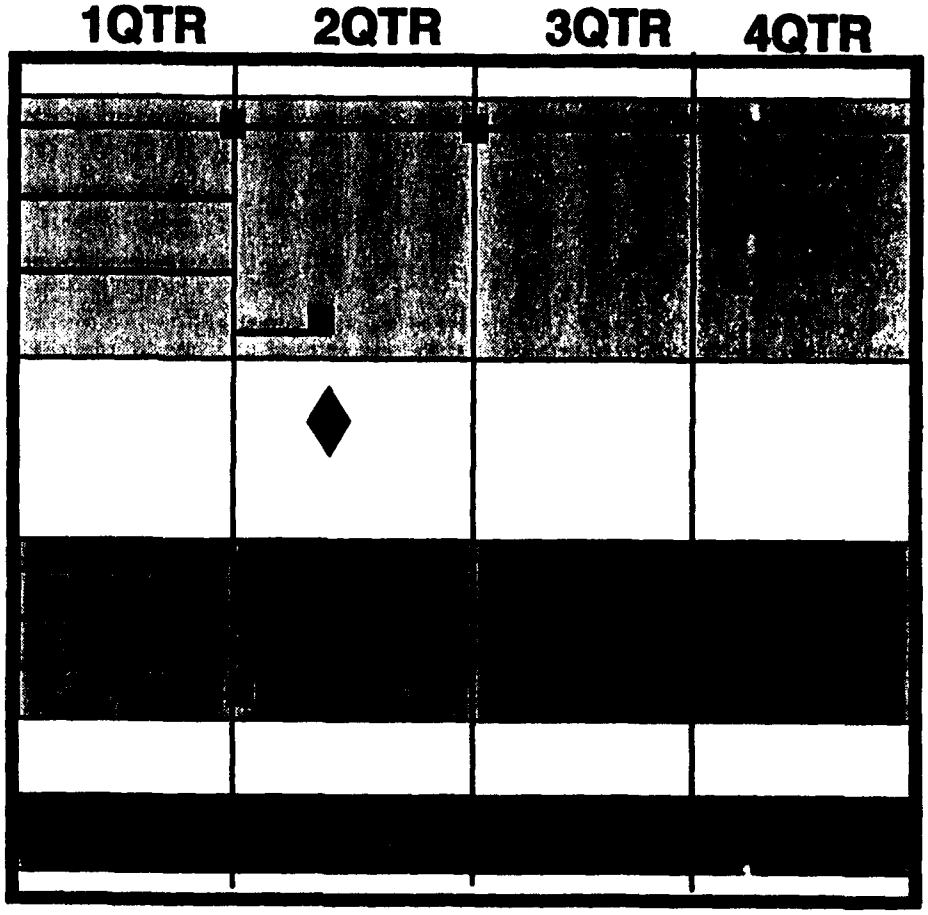
1. Strawman CCSIL
2. Knowledge acq for ground mvr companies
3. Low echelon SAF assessment
4. Technical issues assessment
5. **KA for ground bn**

Task II: Concept/Technology Review

Task III: Implementation Plan

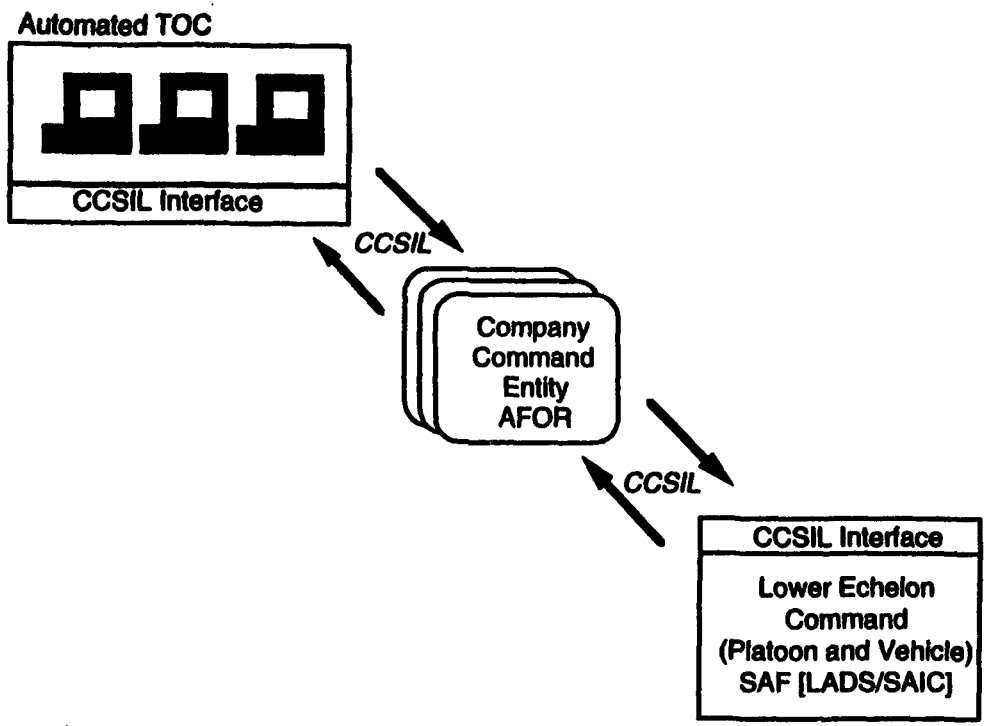
1. Assess MoDSAF SW environment
2. Select CCSIL comm approach
3. Assess lower echelon interface
4. Develop implementation plan

Task IV: Implement Proof-of-Principle Prototype



Level 0 Proof-of-Principle Configuration

AFOR



- **Level 0 POP will assess viability of**
  - **CCSIL as a general purpose vehicle for C3 information exchange**
  - **MODSAF as a software environment for command entity AFOR**
  - **Command entities as a new class of DIS entities**
- **POP configuration will transition for use as an integration and test environment for POC company entities**

Level 0 FY95 - FY97 Activities

- **FY95**
  - **Conduct POP for extension of Technical approach to battalion C3**
  - **Provide integration and test environment for proof-of-concept development with industry developers**
- **FY96-FY97**
  - **Continue integration and test support for proof-of-concept development with industry developers**

Advanced Research Projects Agency

AFOR

Level 1 Proof-of-Concept Development Ground Operations: FY94-97

- **FY94**
  - **Identify industry developers for company command entities**
    - » **Issue BAA early in first quarter**
    - » **Base selection on results of level 0 C2 model definition tasks**
    - » **2 -3 developers on board by end of FY94**
- **FY95**
  - **Develop and demonstrate company command entity**
    - » **Multiple developers; each develops capability for both friendly and OPFOR; down select at end of FY**
    - » **Work closely with designated user organization for knowledge acquisition and experimentation**
  - **Select battalion developers using same process as FY94**

Continued...

Level 1 Proof-of-Concept Development Ground Operations: FY94-97

- **FY96**
  - **Develop/demo battalion ground command entity**
- **FY97**
  - **Integrated Test and refinement of multiple echelons**

Entity and Infrastructure Development

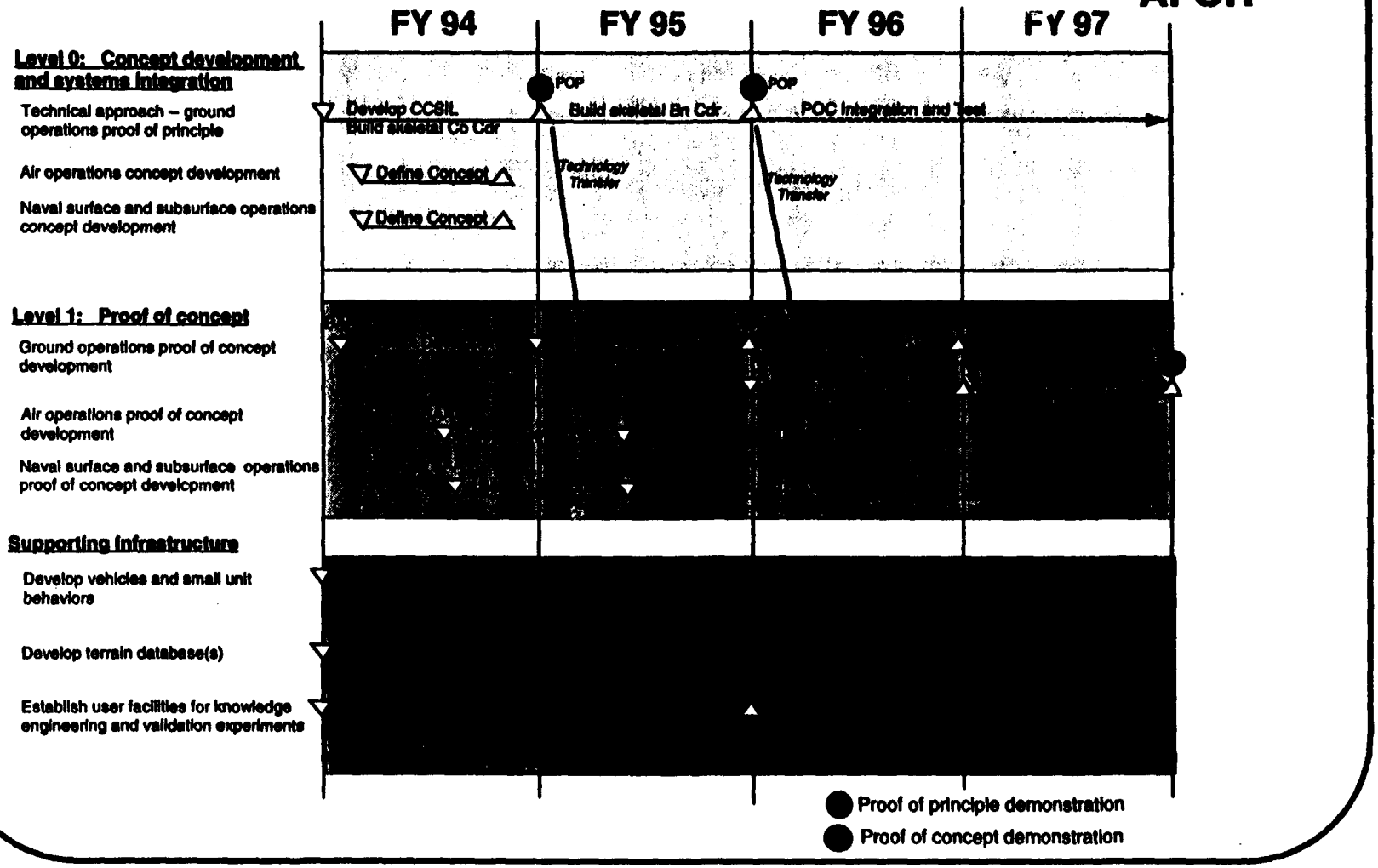
- **Infrastructure requirements for AFOR command entity development**
  - **Small unit commanders and vehicle automated forces will be needed so command entities have entities to command; these include:**
    - » **Dismounted Infantry**
    - » **Marine vehicles**
    - » **Support vehicles**
    - » **Aircraft**
    - » **Fixed targets**
  - **Terrain databases will be needed to support user experimentation and demonstration of technology developments**
  - **User facilities (equivalent to WISSARD site) will be needed to support knowledge acquisition and iterative demonstration throughout proof of concept development**
- **Infrastructure must precede technology experimentation; hence, it must begin early in process**

Transition to STOW 97 Demonstration

- AFOR technology will be transitioned to STOW Development Contractor for integration into STOW 97
  - Working relationship with STOW development contractor needs to be defined
- Key early milestones for coordination with STOW 97 (redirect technology program or STOW expectations)
  - End of FY94
    - » Level 0 proof of principle results for ground operations
  - End of FY95 first proof-of-concept demonstration
- AFOR technology proof-of-concept demonstrations should be tailored to support buildup confidence in the ability to support a proposed STOW 97 scenario

AFOR

FY94-FY97 Schedule



APPENDIX G

SAF CONCURRENT ENGINEERING TEAM BRIEFING (IBM CORPORATION)

SAF Peer Review

October 28, 1993

G-3

CCTT IDT
SAF Concurrent Engineering Team

Agenda

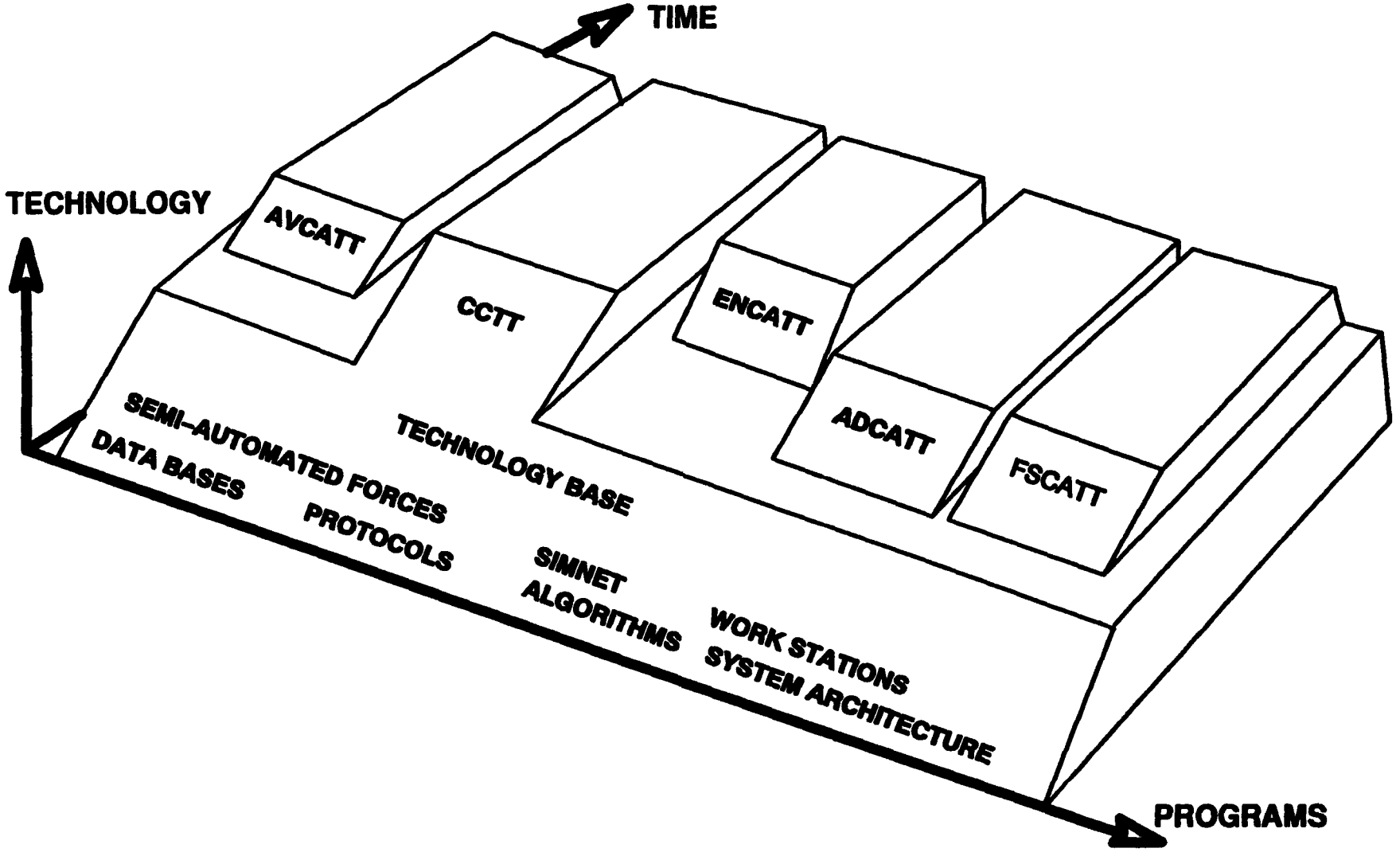
- Overview of CATT and CCTT SAF
- CCTT SAF Technical Rationale
- CCTT CGF Architecture
- Higher–Level Tactical Behaviors in CCTT SAF
- SAF User Computer Interface (UCI)
- Other Topics
- Summary

Overview of CATT & CCTT SAF

- CATT Technology Base
- CCTT Concept of Operation
- CCTT CGF Framework
- Requirements for SAF
- SAF Development Process
- CCTT SAF Development Schedule
- SAF Evolution

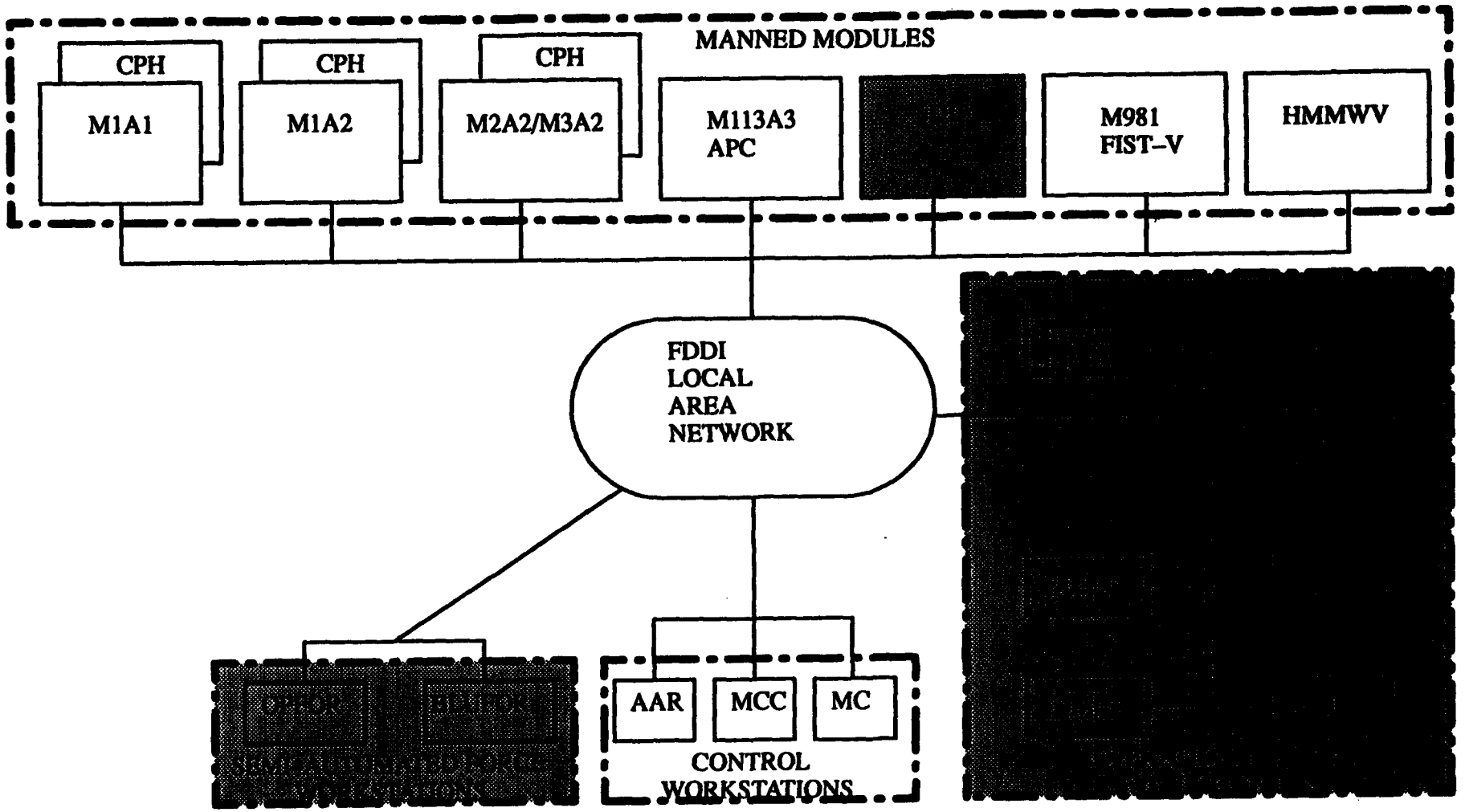
G-5

CCTT Provides the Technology Base for Future CATT Programs



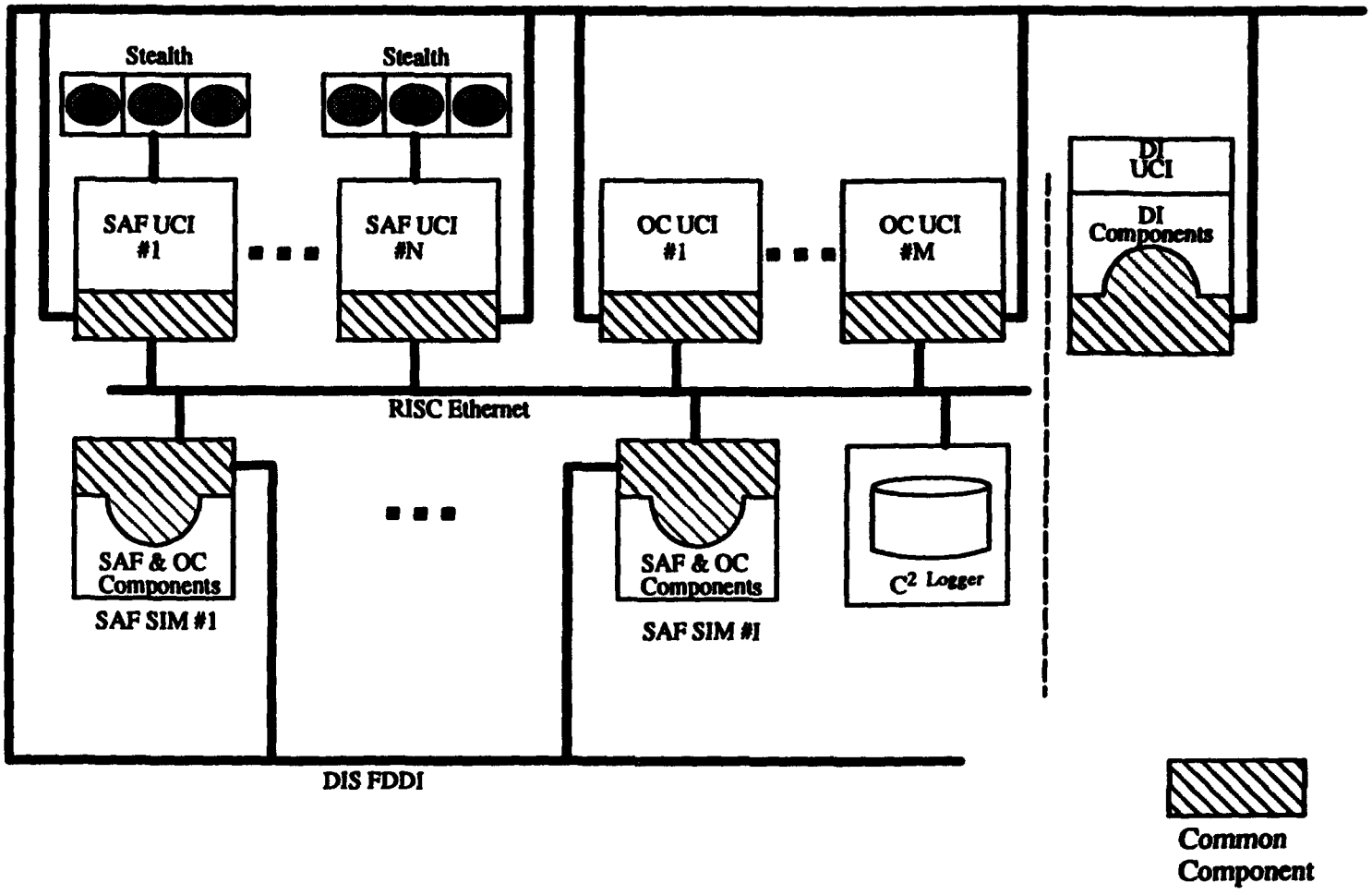
G-6

CGF System Provides all Emulated Entities for the CCTT System



G-7

The CGF Architecture Combines The OC And SAF Functions As One Unit And Provides A Common, Separate Platform For The Manned Modules DI Station



CCTT SAF Requirements Are More Extensive Than for Legacy SAF

Collective Tasks
BLUFOR ~ 700
OPFOR ~ 500

High Fidelity Terrain & Environment
- High Density of Features
- Weather Effects (Rain, Haze, Fog, Cloud)
- Continuous Times of Day
- Tactical Smoke
- Flare Illumination

Dynamic Models

Combined Arms

BLUFOR 53 Systems
A10 Warthog, F15E Eagle, F16 Falcon, M914, M110A2, OH 58D, M2A2/M3A2, AH64 APACHE, German Lec IIMBT, IFV/CFV, French AMX 40 LeClerc MBT, M113A3, DI British Chieftain

MISC 117 Systems
Obstacles, Positions, Ammunition Effects

OPFOR 47 Systems
ASU 85, BTR 70P, SA 6, SU 17, M1975, DI, BM21, HIND-P, BTR 80, BMD II, BMP I, BMP II, MIG 27 Flogger

Simulation Aspects
Obstacle Avoidance
Logistical Effects
Damage Effects
Detect Opposing Platforms
Command From Simulator
Visibility Effects
Jamming
Stochastic Failures

Tactically Realistic
- Traceability of all Entity Models and Unit Behaviors
- Validation, Verification, and Accreditation of all Entity Models and Unit Behaviors

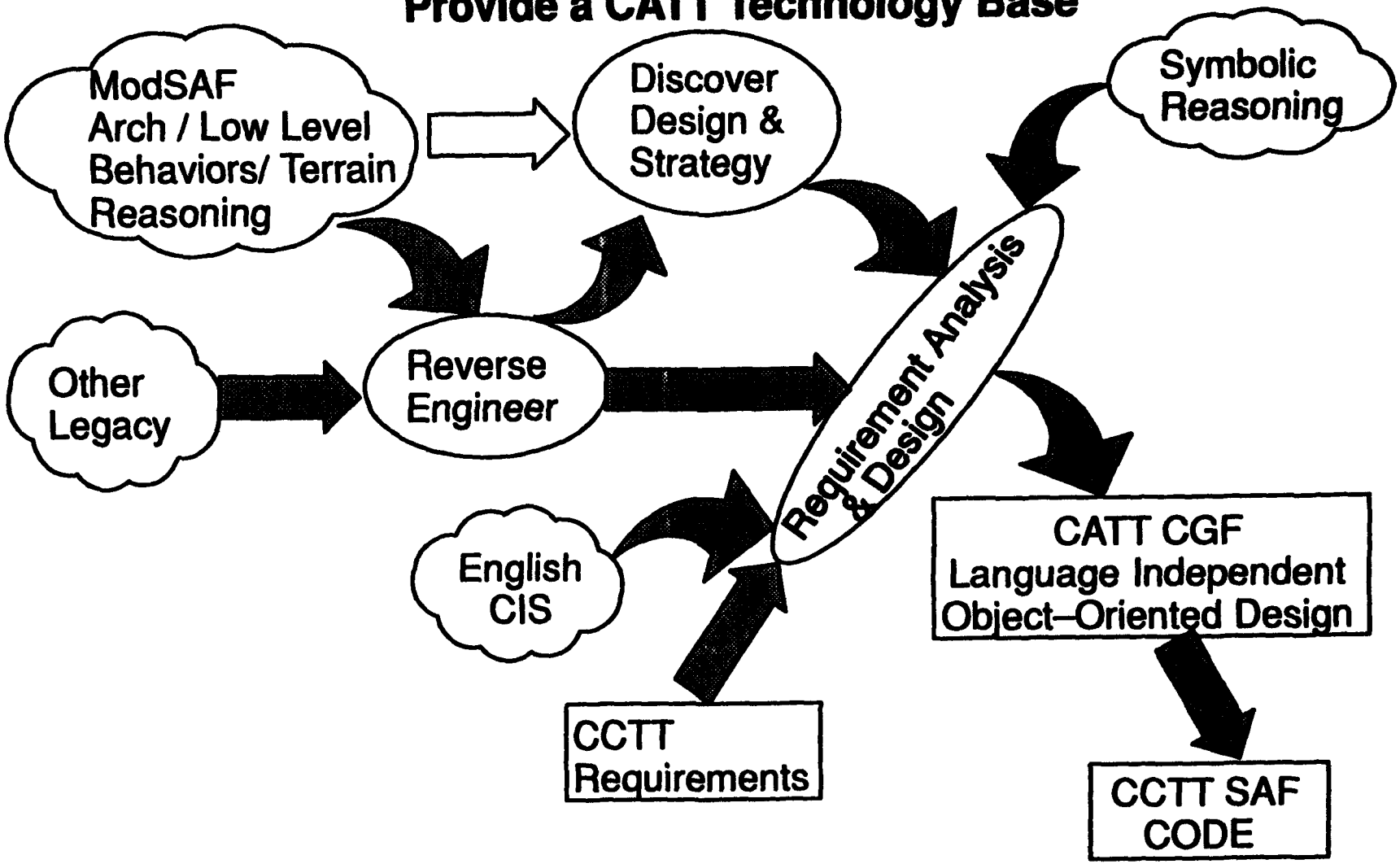
G-9

Requirements For ModSAF 1.0 Legacy Software Are Less Extensive Than Those For CCTT SAF

- Number of Entity Models to be Emulated
  - BLUFOR – 36 Systems
  - OPFOR – 23 Systems
  - MISC – 5 Systems
- Log and Playback of SAF Emulation
- Number of Collective Tasks Implemented
  - BLUFOR – 40
  - OPFOR – 0
- SIMNET Terrain and Fixed Environment Requirements
- Simulation Aspects
  - Obstacle Avoidance
  - Damage Effects
  - Logistical Effects
  - Detect Opposing Platforms
  - Command From Simulator

G-10

CCTT SAF Extends Legacy Software to Provide a CATT Technology Base



G-11

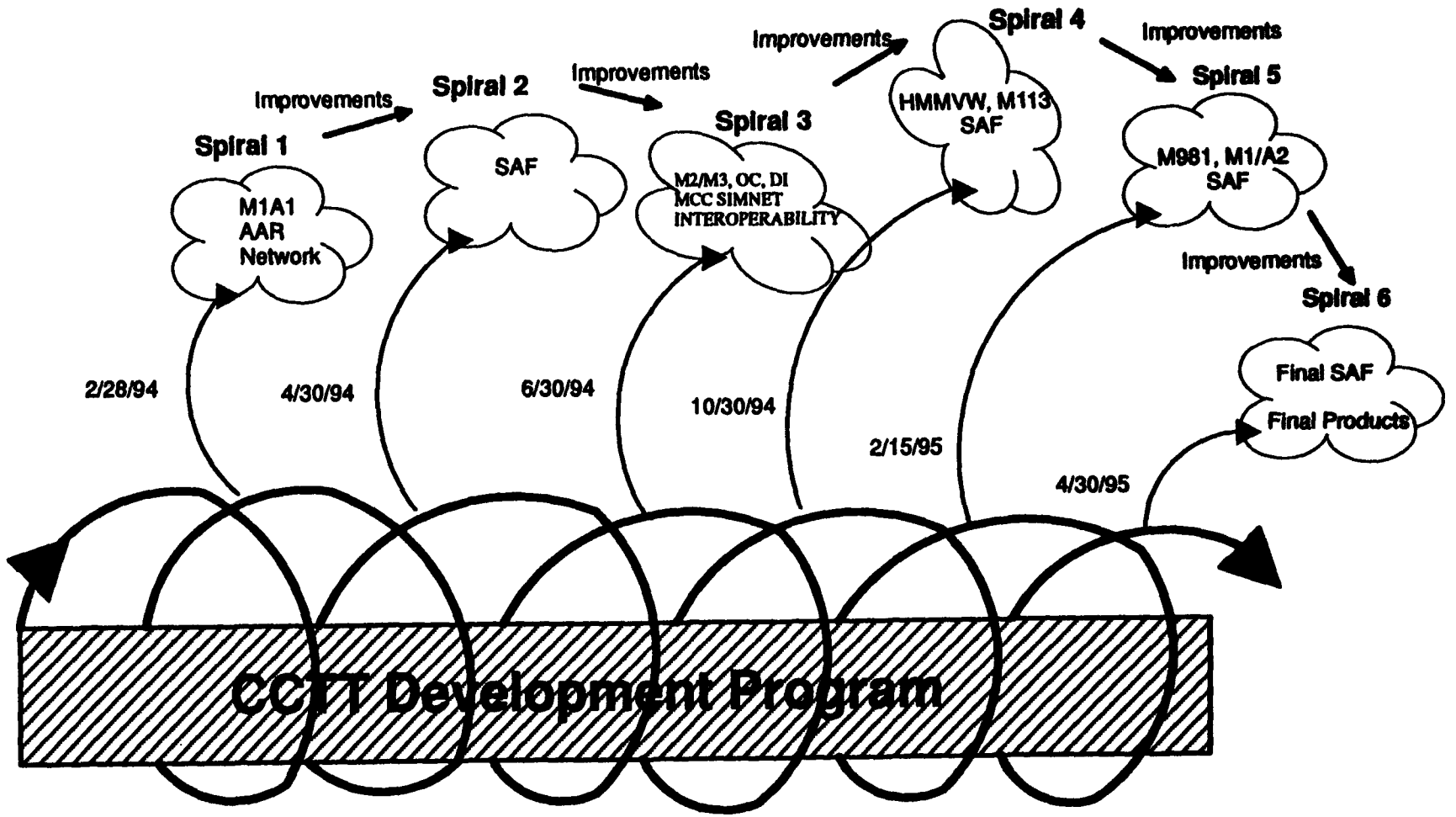
Language Independent SW Design Process

	OOA	OOD	Implementation
Focus	Identify Problem Space objects. Elaborate & define Software Requirements.	Identify solution–space objects. Refine object model. Define interfaces.	Apex => Ada Teamwork => C (? => C++) (Apex => Ada 9X)
Tools	Statemate	Object Maker	
Documentation	SRS/IRS	SDD/IDD	
Keys	Obtain customer/ developer / user understanding and commitment.	Preserve language independence. Trace back to software requirements	

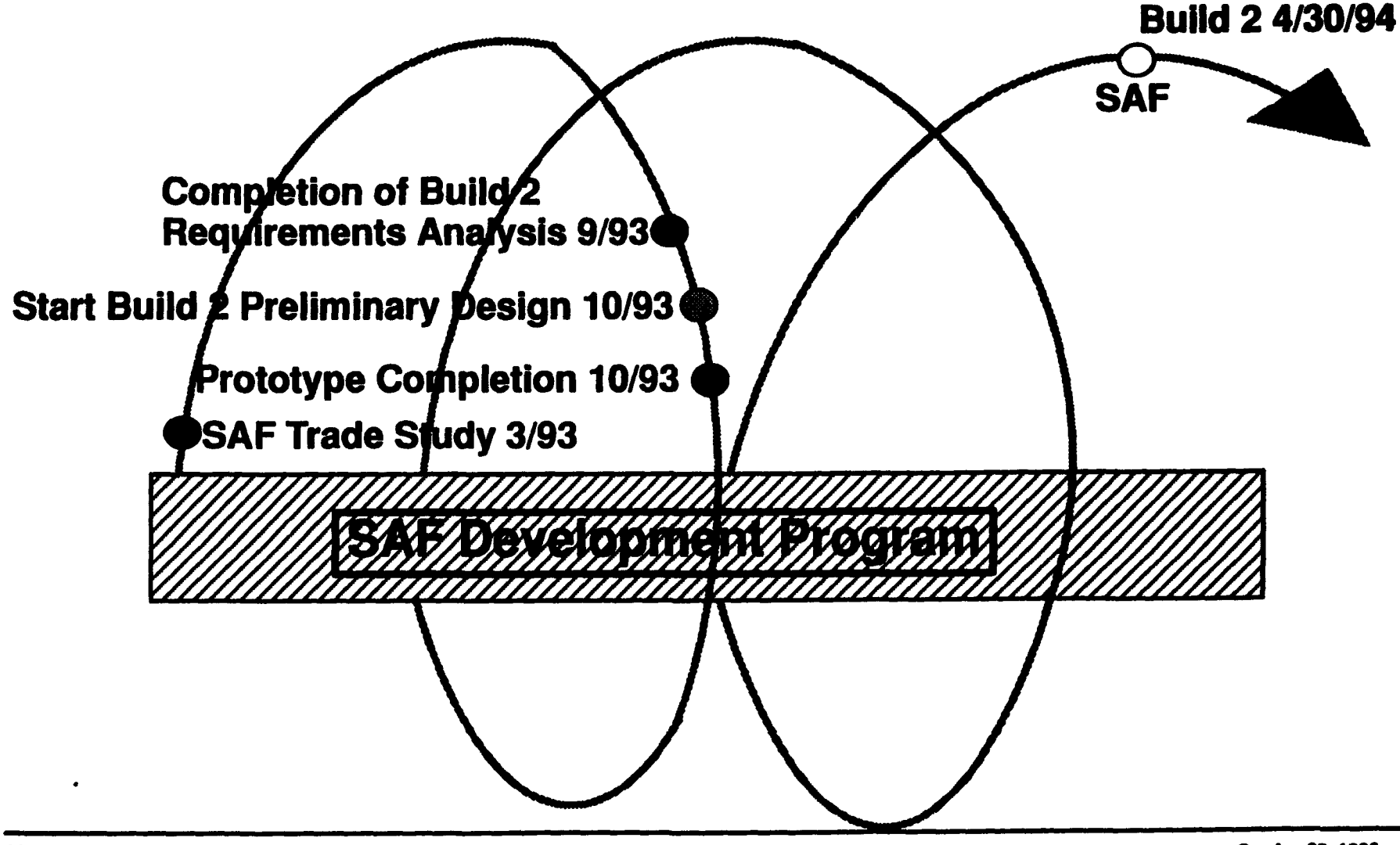
- Life Cycle Support : CIS Editor, Parameter Editor, PM/FL, Network Monitor

G-12

The CCTT Spiral Development Program Provides Iteratively Improved Prototypes for Early Evaluation



The CCTT SAF CE Team Recently Completed Development of SAF Prototypes and Requirements Analysis for the First SAF Delivery

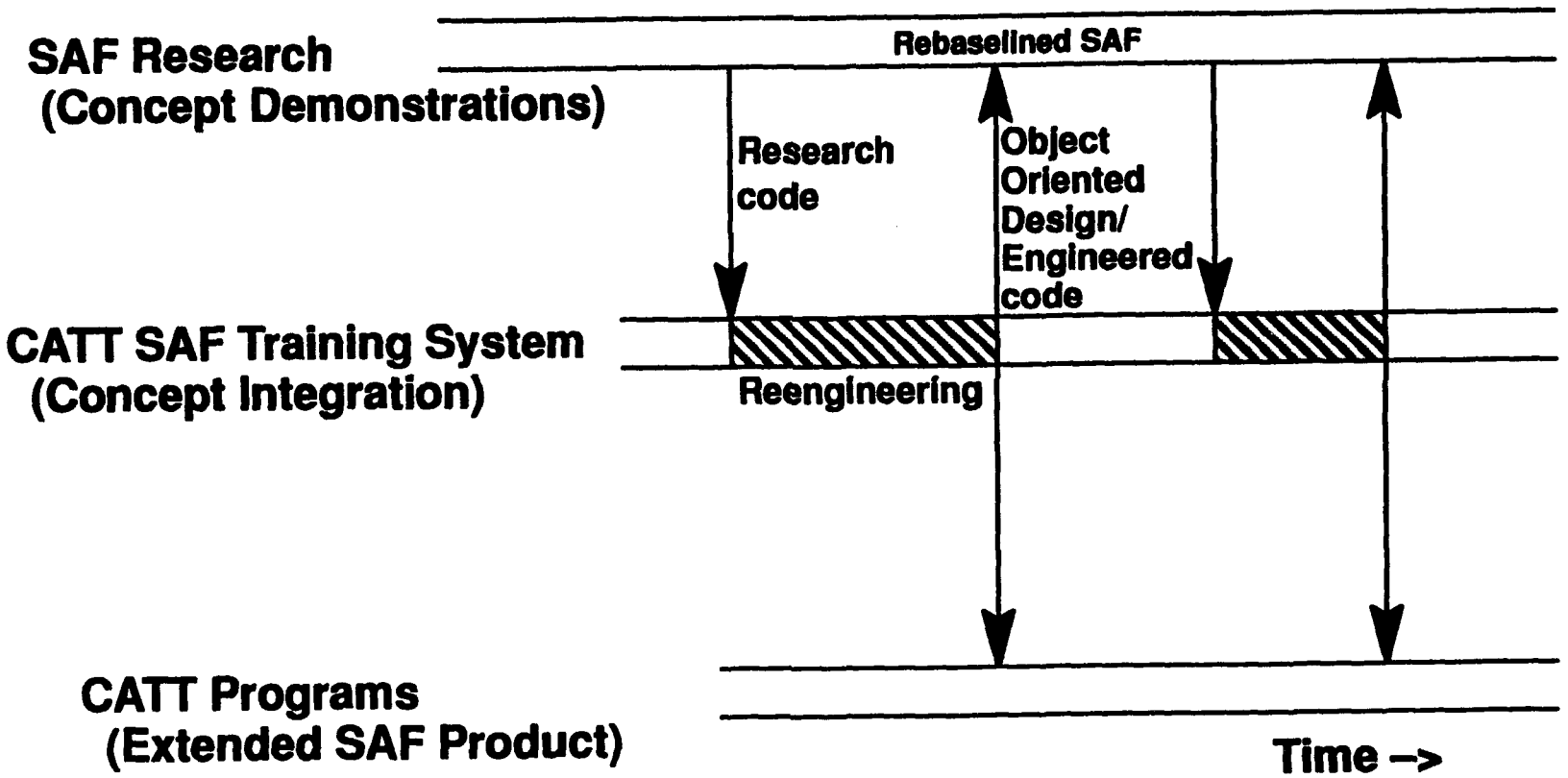


G-14

CCTT SAF Provides an Architectural Baseline Extensible to CATT Programs

- CCTT SAF is being designed to meet present requirements while supporting growth for follow–on CATT SAF
- Architecture and software language supports visibility into the design and facilitates maintenance and incorporation of new functionality
- Finite state machine platform(s) and rule based unit(s) provide efficient means to achieve vertical and horizontal scalability
- TRADOC / operational user inputs and legacy systems that have been user exercised form the foundation of the CCTT SAF development
- Open access to data bases, scenarios, and user inputs facilitates system flexibility and heavy training utilization
- Generation of doctrinally correct combat instruction sets allows play of "school house" default values but retains capability for fine tuning to operational unit SOP and tactics

Reengineering Is the Framework for Continued SAF R&D Evolution



CCTT SAF Provides the R&D Community with a stable Object Oriented Design and Engineered Code baseline using Accredited Tactics and Models

CCTT SAF Technical Rationale

- CCTT SAF Trade Study
- ModSAF Influences on CCTT SAF
- Reengineering Process

The SAF Trade Study Evaluated Four Alternatives to CCTT SAF Implementation

- Alternatives
  - Baseline IDT Approach
  - Reuse ModSAF 1.0 Extended for the CCTT Application
  - Reuse Extended ModSAF 1.0 and RTworks
  - Reengineer ModSAF 1.0 and Integrate IDT Design (Hybrid Approach)
- Categories of Measurement
  - Schedule (Ability to Meet CCTT SAF Milestones)
  - Cost
  - Performance (Ability to Meet CCTT SAF Requirements)
  - Hardware/Software Architecture
  - Software Engineering Standards
    - Extensibility Throughout CATT Program
  - Verification, Validation, and Accreditation

The CCTT SAF Trade Study Evaluated Each Candidate Against Several Criteria

	Sched.	Cost	Perf.	Arch.	Stand. & Ext	VV&A	Total
Base Points	**20**	**20**	**20**	**15**	**15**	**10**	**100**
Baseline IDT Approach	**15**	**9**	**15**	**10**	**12**	**5**	**66**
Reuse ModSAF 1.0 Extended for the CCTT Application	**10**	**10**	**10**	**7**	**10**	**5**	**52**
Reuse Extended ModSAF 1.0 and RTworks	**12**	**10**	**17**	**10**	**12**	**7**	**68**
Reengineer ModSAF 1.0 and Integrate IDT Design	**17**	**9**	**20**	**15**	**12**	**7**	**80**

The SAF Trade Study Recommended Reengineering ModSAF 1.0 and Integrating the IDT SAF Design

- Hybrid Approach is ***Lowest Risk Alternative***
  - Meets All CCTT SAF Schedule and Performance Requirements
  - Creates Extensible Cornerstone SAF for CATT Program
  - Supports Sharing of Reusable Software Components with Other Parts of CCTT and CATT
  - Integrates Finite State Automata (FSA) and Rule–Based Decision Making
    - Best Combination of Representation Technologies for VV&A
  - Facilitates Technology Transfer from ModSAF R&D Efforts
- SAF Trade Study Assumed:
  - ModSAF 1.0 Available September 1993
  - ModSAF A is Upward Compatible to ModSAF 1.0

ModSAF Provides a Starting Point for the Development of CCTT SAF

- Architecture : Basis for CCTT/SAF Architecture Modified To:
  - Incorporate Knowledge Base To Support Tactical Decision Making
  - Provide Object Oriented Design
- Terrain
  - Algorithms
  - Data Structures
- Physical Models
  - Algorithms
  - Structure of Some Models
- Tasks
  - Finite State Machine Generator
  - Algorithms
  - Control Method for Vehicle Behaviors
- UCI
  - Task Interaction Strategies

By Building On A Legacy Foundation We Avoid Reinventing the Wheel and Potential False Starts

G-21

Incremental Deliveries of ModSAF Fit Into the CCTT SAF Development Plan

- ModSAF B
  - Basis of CCTT/SAF Architecture
  - Persistent Object (PO) Database
  - Task Manager
  - Terrain Database
- ModSAF C
  - Tasks and Physical Components for Some Build 2 Vehicles
- ModSAF 1.0
  - Tasks and Physical Components for Additional Vehicles

G-22

Why Not Just Reuse ModSAF?

- Significant Effort Would Be Required to Bring ModSAF 1.0 into Compliance with CCTT SAF Requirements
  - ModSAF Data Models Are not Currently Validated
  - CCTT is a Production Training System vs. R&D Rapid Prototype
  - ModSAF User Interface Software Cannot be Reused
- Software Written in C is Difficult to Reuse in CCTT and CATT
  - Ada/C Language Bindings
  - CASE Development Tools
  - Software Development Standards (Design Documentation, PDL, Coding)
    - Derivation of ModSAF Design Would Still Be Required to Document and Extend For CCTT

G-23

Summary of Benefits of Hybrid Approach

- Generates a Solid Return on Investment in ModSAF
  - Estimated 52,600 Deliverable Ada SLOC Created by Reengineering ModSAF
  - Reengineering Savings Verified Through Metrics Collection & Analysis
  - Effective Transfer of ModSAF Technology into a Production Training Environment
- Establishes a Robust SAF Capability That Can Be Reused in Other CATT Training Devices
  - Ada and 2167A Facilitate Reuse
  - Offers Best Combination of Ada and Rule–based Programming Capabilities for Extensibility

The Hybrid Approach is the Lowest Risk Alternative for CCTT SAF Development

CCTT SAF Reengineering Process

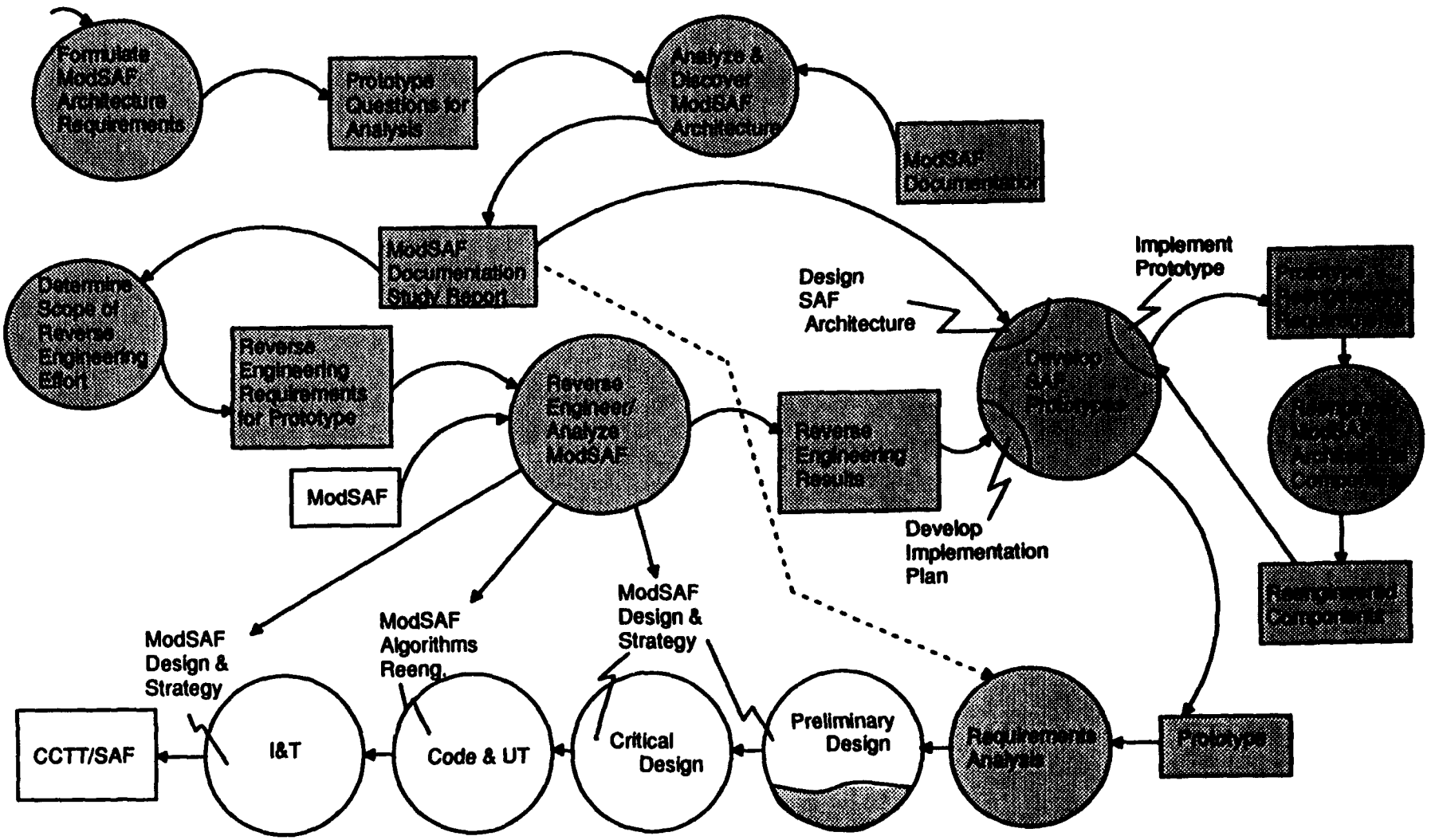
Reengineering Maximizes the Use of Legacy Software Assets

- Reengineering Reduces the Risk of Overall CCTT/SAF Development
  - Exploits Proven Legacy Software
  - Only Relevant Parts are Selectively Reengineered
- Facilitates Transfer of Research Technology into a Production Training Environment
- CCTT/SAF Reengineering Process is Integrated Into CCTT Development Process
  - Produces a Language–Independent Design that Captures and Improves on the ModSAF Approach
  - Maximizes the Benefit Obtained from ModSAF During Design/Coding Phases
- Resulting CCTT/SAF Design Components Available For Other CATT Programs

G-26

Reengineering Provides a Production Quality SAF Meeting CCTT Needs and Facilitates Future Reuse of Research Innovations for CCTT

CCTT/Reengineering Process Overview



Shading Indicates Completed Activities

G-27

CCTT CGF Architecture

- Overview
- Architecture Prototype
- Terrain Prototype
- SAF Language Issues

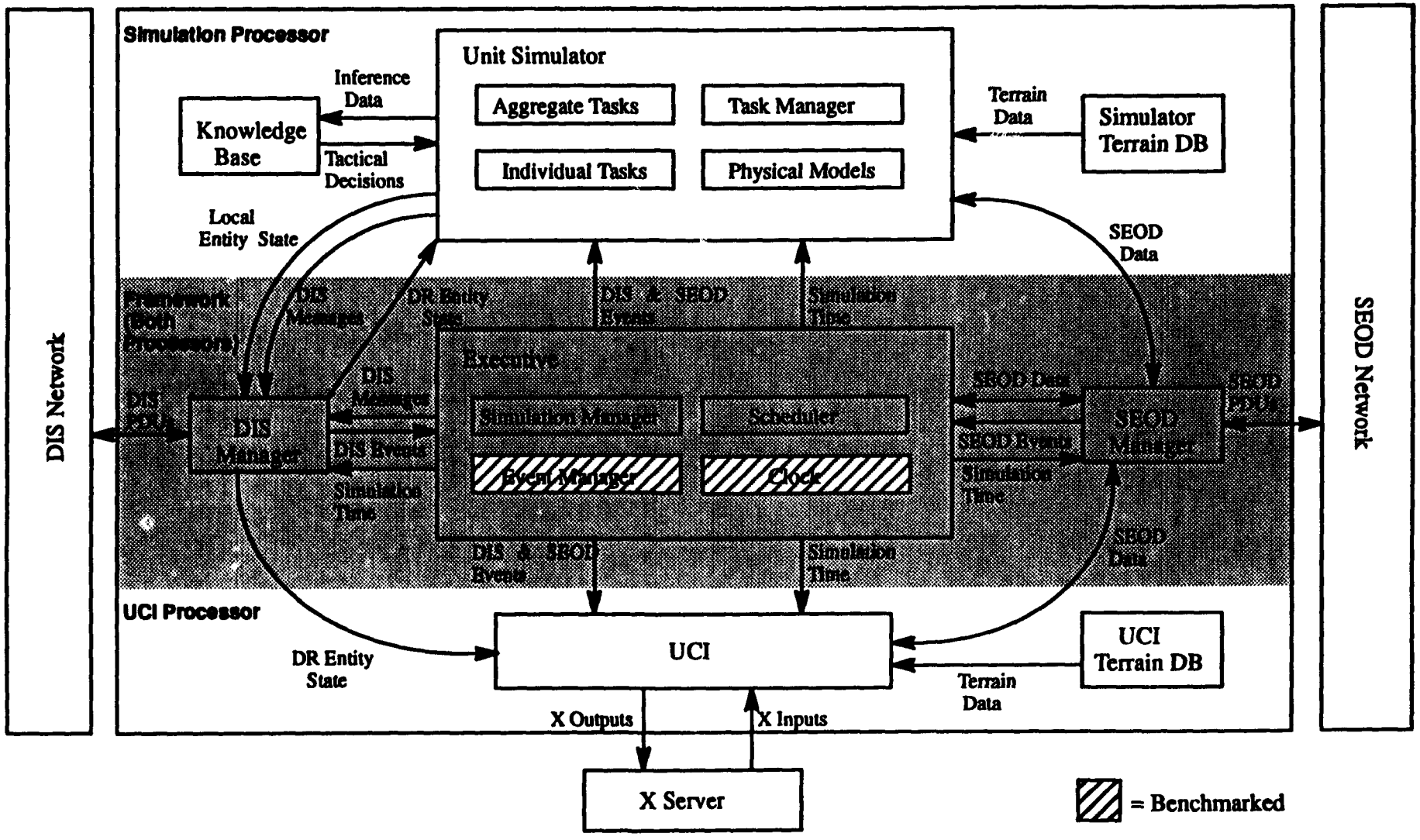
G-28

The CGF Architecture Extends ModSAF Concepts to Implement a Seamless Environment for SAF, OC, and DI Components

- Uses the ModSAF SAFsim and SAFstation approach
- Explicit CGF command and control is implemented across the network in the SAF Entity Object Database (SEOD)
  - The SEOD is an extension of ModSAF's Persistent Object (PO) database concept
  - The extension supports symbolic reasoning of collective entities, ownership of platforms by workstations, and additional CCTT system requirements
- Modular, extensible design enables Workstations CE Team to construct models for Operations Center vehicles
  - Vehicle composition is entirely data–driven
- Each computer system contains all the software necessary for the emulation of CGF entities (there are no dedicated servers for CGF terrain or inference engines)
- Operations Centers and Dismounted Infantry become SAF stations with different UCI components

The CGF Architecture provides a modular, reusable baseline supporting horizontal and vertical extensibility.

The SAF Framework Component Provides Common Capabilities for UCI Processors and Simulation Processors

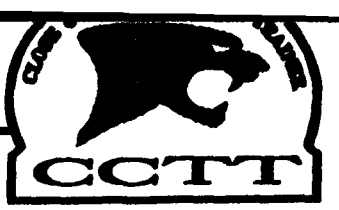


The SEOD Approach Supports Key Features of Emergent SAF Systems

- Extensibility: Additional computer systems may be added to extend the number of platforms simulated at a site.
- Command and Control: Platform control by a UCI or a simulated unit commander is obtained through the SEOD protocol.
- Load balancing: Automatic load balancing makes for efficient use of the computer resources maintaining the fidelity of simulation.
- Migration of vehicle simulation: Platform simulations may be moved between machines on an as needed basis.
- Fault tolerance: Platforms simulated on a failed computer system will be redistributed between the existing systems.
- Reset: Sharing of SAF C2 information facilitates SAF logging, checkpointing, and reset capabilities.

This provides scalability of the number of simulated entities per site, reduced single point failures, and flexibility in allocation of computer resources to exercises.

Architecture Prototype



The SAF Architecture Prototype Developed an Architecture for CCTT SAF Which Builds Upon the ModSAF Legacy

- Analyze the ModSAF Architecture, as Represented by ModSAF B
  - Apply Reverse Engineering Tools to Derive the ModSAF Design
  - Consult with Loral ModSAF Expert in Orlando
- Develop an Architecture for CCTT SAF
  - Focus on Procedural Software Framework
  - UCI, Knowledge Base, and Terrain Addressed in More Detail by Other Prototypes
  - Retain Essence of ModSAF Design While Meeting CCTT System Requirements/Objectives
- Experiment with Ada Implementations of "Difficult" Parts of ModSAF Design
  - Event Generation and Processing
  - Component Structure, Including Generic Model Interfaces

The guiding principle was to produce a open, scalable architecture built from modular, reusable components.

G-34

Static Definition of Event Types Gives Event Manager a Performance Edge over ModSAF LibCallback

- Operation
  - Generate ("Fire") an Event
- Scenario
  - Event has 1 Integer, 1 Pointer, plus User Data
  - 3 Event Handlers (Null)
  - IBM RISC System/6000 Model 530H
  - Optimized
- Measurement Results
  - Event Manager (Ada) : 3.60 uS
  - LibCallback (C) : 5.34 uS
- Analysis
  - Performance Factor from C to Ada is 0.67
  - LibCallback Interprets Event Function Profile at Run Time

G-35

Measurements Show Interfacing Directly to System Calls is Almost 3 Times Faster Than Using Package Calendar

- Operation
  - Advance Simulation Clock
- Scenario
  - IBM RISC System/6000 Model 530H
  - Optimized
- Measurement Results
  - Clock (Ada) : 232.40 uS
  - LibTime (C) : 81.50 uS
- Analysis
  - Performance Factor from C to Ada is 2.85
  - Vendor Implementation of Package Calendar Computes Absolute Time
  - Option: Interface Directly to System Calls (as in C)

G-36

The SAF CE Team Developed a Baseline Architecture that Meets CCTT SAF Requirements and Provides Commonality for Research Organizations

- ModSAF Analysis
  - Maximizes Reengineering Benefits
  - Prevents Accidental Deviations from ModSAF Design that Would Inhibit Reengineering
  - Facilitates Technical Interchange with Research Community
- Architecture
  - Selected Architectural Components were Implemented and Object Diagrams were Developed
  - Provides Good Starting Point for Preliminary Design
- Ada Lessons Learned
  - Ada is an Effective Implementation Language for the Proposed ModSAF–Based Design

G-37

Terrain Prototype

The Goal of the Terrain Prototype was to Identify a Direction for CCTT SAF Terrain Processing Based on Leveraging Industry Approaches

- Compare and Contrast Industry Algorithms
- Identify Requirements on Terrain Algorithms and Data in CCTT SAF
- Develop an Approach for Building the SAF Terrain Databases that Guarantees Correlation Within CCTT
- Characterize the Effects of Implementation Language on Timing

G-40

The SAF CE Team Chose a Portion of the Intervisibility Algorithm from ModSAF and Implemented Test Versions in Both C and Ada

- Intervisibility Was Chosen Because it Consumes a Large Fraction of the Total CPU Time and Presents a Worst–Case Scenario for Ada Performance
  - Rest of Terrain Processing Would not be as Harsh on Ada
- Tests Included Vehicle Blockage, Terrain Blockage, Then Both.
- Constructed a Controlled Test Environment
  - 5 Database Representations: 2 Hand Made, 3 Random, 2 Km x 2Km in Size
  - 50 Vehicles: 2 Hand Made, 3 Random Placement
  - Intervisibility Calculations Between Each Vehicle
  - Concerned with Comparison of C and Ada Representations
- Built Three Versions of the Test Algorithm: C Version, Ada Translation, and Optimized Ada.
- Baseline Test Removed Initialization Time from the Run–time Tests

G-41

ModSAF Approach Provides an Excellent Starting Point for the Development of CCTT SAF Terrain Processing Capabilities

- Final Metrics Demonstrated that Ada Performance Was ≈1.35 w/checks 1.08 Times C with Run–time Checks Disabled
- This Provides an Excellent Basis for CCTT SAF Algorithms and Approaches
- Terrain Algorithms Can Be Done in Ada; However Selected Portions Need to be Optimized to the Language
- Classically, Ada's Performance Penalties Can Be Traced to Automatic Run–time Checking
- CTDB and Quad Data Representations Can Be Extended to Satisfy CCTT SAF Requirements

G-42

This approach supports extensibility and inclusion of additional algorithms from ModSAF research.

SAF Language Issues

Language Choice Reflects More Than Just Technical Decisions

C is:

- Language "of choice" of research community
- Good for "small projects" (< 100k SLOC)
- Prolific in human resources
- Heavily oriented toward performance
- Development costs reduced at expense of life cycle costs

Ada is:

- Language "by mandate" of development community
- Good for "large projects" (> 100k SLOC)
- Limited, but growing, in human resources
- Cost efficient for project life cycle
  - Non–compact structure promotes understandability
  - Formal Qualification Test (FQT) statistics indicate Ada has fewer errors than C code
- More oriented toward error checking than performance

Large projects typically benefit from the software engineering principles implemented in the Ada language since the language was developed specifically for large programming projects.

The SAF CE Team Can Render CCTT/CATT SAF in Either C or Ada

- Choice of language is a delicate balance between technical and programmatic issues
- Different language efficiencies should drive the implementation only, not the design
- Implementation approaches reflect the constructs and techniques available to the language
- The SAF CE Team recommends using Ada
  - Performance can be improved by optimization techniques, algorithmic changes, and growing technologies
  - Performance is a trade for portability, reliability, understandability, and future maintenance
  - Ada facilitates integration, maintenance, and FQT activities
  - CCTT/CATT benefits directly by CCTT reusable components for development and correlation
- CCTT/CATT will maintain 2167A process independent of language to support traceability, visibility, maintenance

The use of Ada directly promotes present and future reusability, controlled configurations, and life cycle maintainability.

Higher–Level Tactical Behaviors in CCTT SAF

- SAF Tactical Behaviors (CIS) Development Process
- SAF Tactical Behaviors Code Implementation
- SAF Tactical Behaviors Prototype
- Value Added by CCTT SAF Behaviors

SAF Combat Instruction Set (CIS) Development Results in Validated Tactical Behaviors Traceable to Doctrine

- CISs defined as combat behavior collective tasks
  - Represented as English language descriptions
  - Structure defined by IDT subject matter experts, knowledge engineers, software engineers and Army TSM
  - Validated by Army subject matter experts
    - BLUFOR CISs validated by TSM–CATT
    - OPFOR CISs validated by CAC–Threats
- Validated CISs for unit tactical behaviors are developed, documented and traced to code
- Validated, verified and approved CISs are stored in RCI database
- Both US and opposing threat tactics are explicitly represented

Validated CISs Provide Source Documentation for Implemented SAF Behaviors and Serve as Their Design Specifications

G-48

SAF Database Ensures Easy Access to Code, Doctrine and CISs in an Integrated Traceability Network

- The CCTT SAF traceability database will identify:
  - The ARTEP/doctrinal reference for SAF code
  - CATT task source for each CIS implemented in SAF code
  - CIS–to–code functional traceability table
  - SAF source code reference
- A unique database, table, or text file will be defined, populated, and maintained for each of the following:
  - ARTEPs/FMs used in SAF
  - CATT tasks (via RCI)
  - English language CIS
  - SRS and PIDS
  - CSCIs
- SAF's functional traceability database is defined within the context of the CCTT traceability database

The SAF Database Ensures Capture of Design Rationale and SAF Community Access to All Supporting Documentation

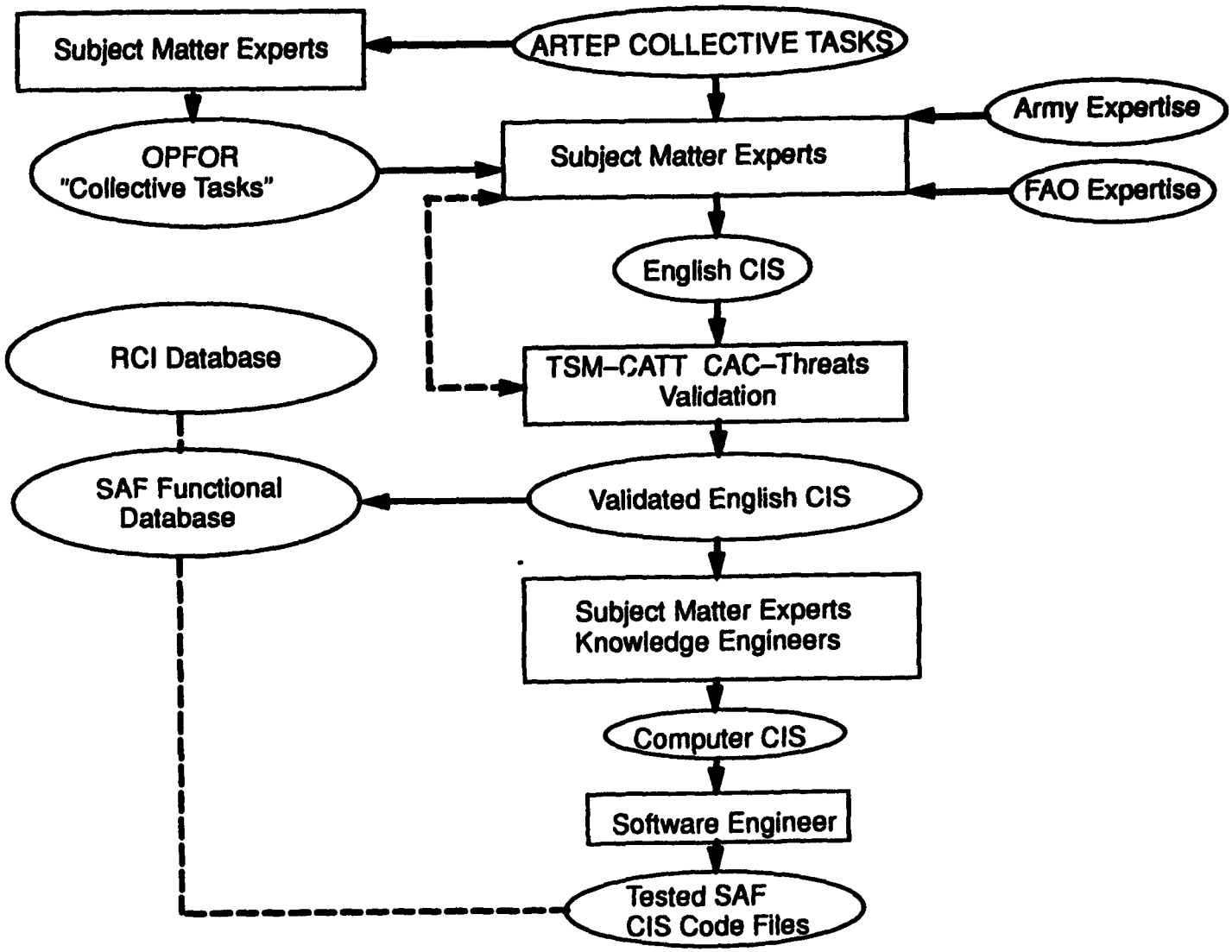
The Scope of SAF CIS Development Requires a Formal, Approved Process

- Requirements based on Table A–1 of the CCTT Prime Item Development Specification
- Approximately 700 BLUFOR collective tasks
- Approximately 500 OPFOR collective tasks

G-50

A Very Large Number of VV&Aed Combat Behaviors are Required above the Platform Level to Support CCTT and CATT

CIS Development is a Formal Process Approved by the Army



G-51

BLUFOR CIS Development Status

- Produced to date
  - 43 Tank Platoon CISs
  - 22 Scout Platoon CISs
- Current efforts
  - Restructured 43 Tank and 17 Scout Platoon CISs
  - Drafted 28 of 52 BFV Platoon CISs
  - Development underway to provide sufficient combined arms CISs to understand company team. This includes:
    - Mortar Platoon
    - FA Cannon Firing Platoon
    - Combat Engineer Platoon
    - ADA Platoon/Team (STINGER/BSFV)
    - Antiarmor CO/PLT/SEC

G-52

OPFOR CIS Development Status

- Produced to date
  - 24 Tank Platoon
  - 23 Motorized Rifle Platoon
  - 23 Reconnaissance Patrol/Platoon
  - 8 Antitank Squad
  - 9 Antitank Platoon
  - 6 of 12 Self–propelled Artillery Battery
- Current Efforts
  - Restructure/Fix Tank Platoon based on CAC–Threats Comments/TSM format changes
  - Begin work on Tank Company

G-53

SAF Behaviors Code Implementation is an Integral Part of the CIS Development Process

- Implementation of SAF tactical behaviors in rules is executed concurrently with the CIS Development Process
- Tactical behaviors code implemented as modular, realtime rulesets
- Code documentation includes traceability to CIS and doctrinal literature
- Code file names directly reflect CIS names, facilitating maintenance, extensibility and navigation through code files
- Code structure closely resembles CIS structure
- Code development governed by well–defined IDT process involving:
  - Structured programming techniques and documentation
  - Configuration management and QA
  - Version control
  - Unit test, integration test, and continuous code reviews

Close Mapping Between English Language CISs and Code Facilitates Strong Software Development Practices

Resultant SAF Code is Vertically and Horizontally Scalable

- Classes and objects used to define SAF vehicles, units, & systems allow extensibility through inheritance
- Tailorable tactical behavior modules capture unit–level decision making above the platform level
  - Small behaviors modules can be flexibly assembled to tailor a set of combat behaviors for a specific exercise
  - Simplified training behaviors code can be exchanged with more complex code required for analysis or R&D
  - New unit behaviors can be added at any echelon defined for CCTT SAF by including them in the exercise application file
  - Existing tactical behaviors modules may be omitted from an exercise in similar manner to reduce needless processing
- Well–specified interface design will allow platform modules to communicate with unit behaviors in an open architecture
  - Interfaces defined for information flow between platforms & platoons
  - Interface design is well–controlled and configuration managed

CCTT SAF Open Architecture Provides Flexibility in Assembling Code Modules to Fit the Needs of An Exercise or Application

G-55

SAF Behaviors Code Captures the Nature of Tactical Reasoning Above the Platform Level

- SAF software developed from CISs exhibits doctrinally correct tactical behavior that is:
  - Traceable to US and OPFOR doctrinal literature
  - Tailorable to new BLUFOR and threat tactics via behavior module substitution
  - Extensible within and among echelons by adding or modifying behavior modules
  - Approach applicable to combined arms tactical behaviors
- Rules map closely to the English language CISs
  - Easier for engineers to translate CISs into rules
  - Easier for SMEs to review and validate code

G-56

SAF Behaviors Code Maps Closely to Tactical Knowledge, Facilitating Knowledge Transfer from SMEs to Software Developers

SAF Behaviors Code Captures the Nature of Tactical Reasoning Above the Platform Level (cont.)

- Rules naturally capture multi–contingency tactical reasoning
  - Eliminates need to predefine & precompile every tactical response
  - Tactical decision–making tends to be heuristic & symbolic in nature
  - Critical tactical decisions tend to be event–driven and asynchronous
  - Situational awareness involves evaluating among many alternatives in a non–deterministic way
  - Rules permit sophisticated arbitration
- Rules can be generalized to high levels in the class hierarchy for generic reasoning or specilized to low levels for reasoning
- Rules are used to represent reactive behaviors that do not have to be preplanned in the operations order–they are event & situation–activated
- CCTT SAF uses two control mechanisms, each where it is most appropriate–without limiting developers/researchers to either one

SAF Code can be Generalized for Reuse of Common Behaviors or Specialized to Achieve Systems–Specific Behaviors

G-57

SAF Behaviors Arbitration is Based on Maintained COTS Product

- COTS inference engine–RTworks–used as tactical arbitration mechanism
- Multiple arbitration mechanisms will be supported
- Rulesets only execute when conditions are met–an apt environment for asynchronous tactical decision–making
- Real–time rulesets no longer suffer from Lisp's garbage collection delays
- Inference engine is a separate process integrated through an IPC mechanism with carefully defined protocol
- Rules are grouped into small, modular rulesets that define a unit CIS or closely related set of CISs

G-58

Use of COTS Inference Engine Provides Powerful Arbitration Mechanism, Modularity of Behaviors, and Real–Time Performance

CCTT SAF Behaviors Prototype Demonstrates Feasibility of CIS Approach

- Used English language CISs as specification for BLUFOR/OPFOR tank platoon collective task behaviors
- Implemented BLUFOR tank platoon CISs as structured rulesets as first test case
  - 25 CISs prototyped fully; 18 others less extensively
  - Focused on platoon tactical decision–making
  - Rulesets structured as tactical behavior modules that are easily modified, exchanged, or omitted in tailoring an exercise
- Captured situational awareness component of CISs in prototype
  - Tactically significant events modeled as situational interrupts
  - Situational interrupts (SIs) trigger battle drill CIS responses
  - SIs interrupt an operations order without corrupting it
  - Multiple contingencies handled as nested, prioritized SIs

G-59

The CIS Development and Implementation Process has Allowed SMEs to Effectively Transfer Tactical Knowledge to Software Engineers

CCTT SAF Behaviors Prototype Demonstrates Integration of Three Types of CISs

- Prototyped tank platoon operations order as a sequence of CISs triggered by time, control measures, situations or combinations
- Prototyped command override capability for all 43 CISs
- Developed CIS prototyping environment designed to facilitate spiral development
    - Easy test case data and test scenario generation
    - Viewing of op order execution plan (simulating paragraph 3)
    - Viewing of battlefield events as they occur during simulation
    - Viewing of platoon combat behavior history for testing/AAR support

G-60

CCTT SAF Behaviors Design Facititates Integration of Operations Orders, Ad Hoc Command Overrides, and Reactions to Situational Interrupts in a Natural Way

CCTT SAF Behaviors Prototype Operations Concept Provides a Solid Basis for Design

- Emulated data flow between platforms and platoon behaviors
- Emulated data flow between UCI and platoon behaviors
- Multiple demos performed for IDT Peer Review, COL Shiflett, MAJ Johnson, H. Marshall, MAJ Briggs, M. Petty, D. Mullaly, SAF Working Group: October 14–25
- Performed initial performance benchmarking: report in progress
- Using IDT coding standards document CM, QA, documentation, and version control processes

G-61

Lessons Learned from the SAF Behaviors Prototype will Drive the Design of Tactical Behaviors Code

Value Added SAF Behaviors Development and Implementation Process Adds Substantial Value to Legacy SAF Code

- Validation of combat behavior by Army schools
- Traceability to Army training documents and doctrinal literature
- Formal CIS development and VV&A process
- SAF code that is:
  - Doctrinally correct
  - Formally tested and reviewed
  - Traceable to US and foreign doctrine
  - Vertically and horizontally extensible
  - Engineered for extensibility, reusability, and maintenance
  - Based upon proof–of–concept prototyping

G-62

CCTT SAF Behaviors Code Provides a C2 Tactical Umbrella for SAF Platforms Code Resulting from a Rigorous, Repeatable Knowledge Capture, Transfer and Translation Process

User Computer Interface (UCI)

- Development Strategy
- UCI Military Symbology
- CCTT SAF Operating Concepts
- CCTT SAF UCI Editors

The CCTT SAF UCI Development Strategy Will Ensure a User Friendly, Extensible System

- User (TRADOC) involvement from the start
  - Current/Former SAF Operators which are part of the IDT have provided early feedback through the rapid prototype development
- The SAF UCI Team is performing an analysis of SAF Legacy systems and the tasks of their users
- The SAF UCI is consistent within CCTT and other applications
  - Designed in accordance to CCTT Design Guide which is based on OSF/Motif Style Guide and customized to CCTT needs
- Use of the OSF/Motif User Interface Language (UIL) and GUI Builder Tools promotes extensibility
  - UIL is a component of OSF/Motif, 100% portable, language independent, and is Imported and Exported by most GUI Builder Tools today
  - The use of GUI Builder tools reduces the amount of software development for the User Interface
- The SAF User Interface facilitates the preparation of military overlays and the creation of exercise scenario libraries thereby allowing exercise conditions to be replicated and analyzed

G-64

The SAF UCI Utilizes Military Terms, Symbols, and Concepts

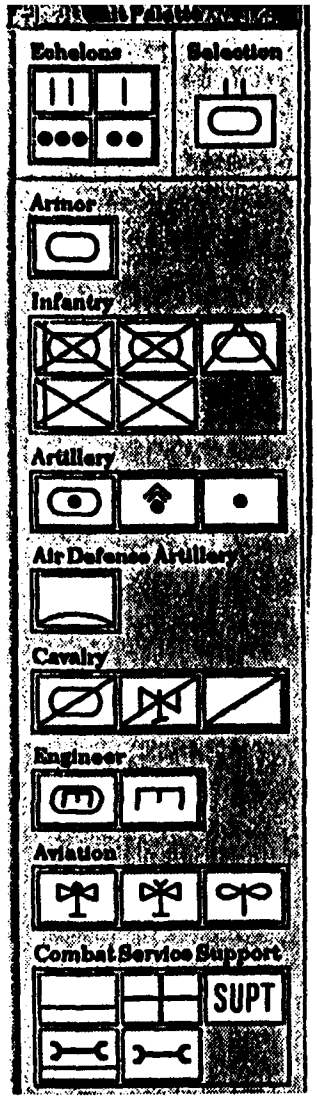
- FM 101–5–1 "Operational Terms and Symbols"
- Selected OPFOR symbology
- Icon symbols highly used instead of text
- Overlays created analogous to military operations graphics
- Exercises build in similar sequence as an Operations Order (OPORD)
- Use of Military terms instead of software terms (Overlays vs. Files)

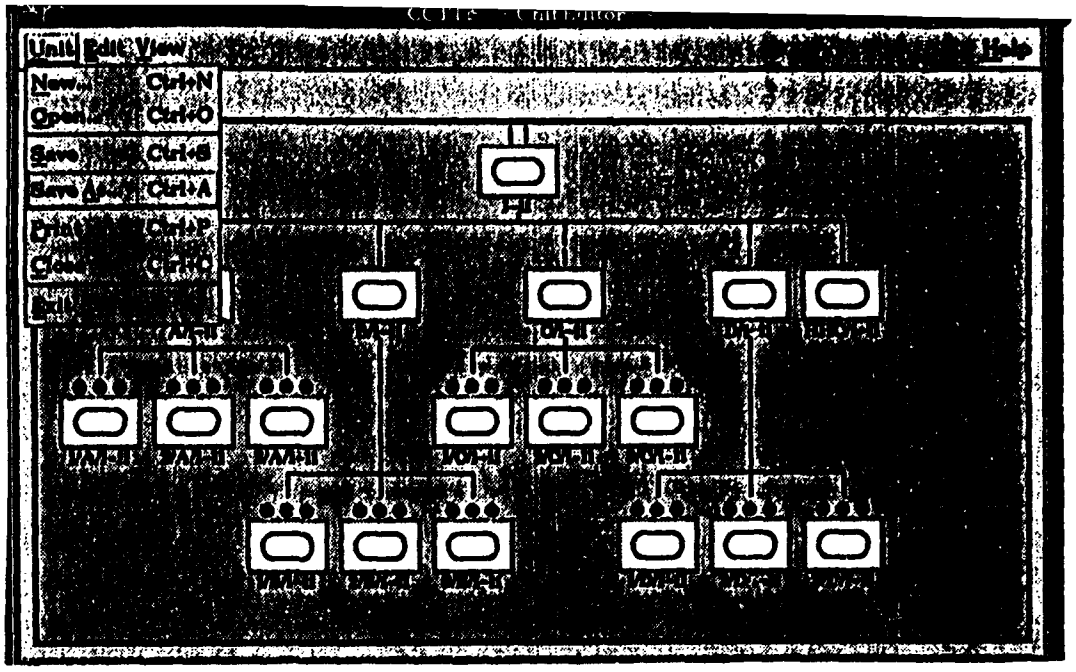
G-65

The CCTT SAF User Interface is Tailored to the Different Types Of Tasks Performed By the SAF Users

- Pre–Exercise
  - Functionality focused on Editors to task organize units, create overlays, and build exercise scenario similar to an OPORD
  - User sets up all information necessary for the exercise on a stable environment
- Exercise Execution
  - Functionality focused on the control of the SAF exercise units and the display of the battle status
- Offline
  - Allow Parameters to be easily added and modified
  - Supports Software Full Life Cycle
  - Tools identified and required to provide offline support

G-66


Echelons
Selection
Armor
Infantry
Artillery
Air Defense Artillery
Cavalry
Engineer
Aviation
Combat Service Support
SUPT


Unit
Edit
View
Help
New... Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As... Ctrl+A

SAF Controller

64

The SAF UCI Editors are Engineered Using Formal Style Guide Standards that Ensure Consistency, Enhance Understanding, and Promote Ease of Use

- Cohesive
  - Set of editors allow the user to task organize units, build military overlays, and create exercise scenarios independently
  - Allow the users to build exercise component libraries
  - Exercise scenarios are easily generated by selecting predefined components
- Data driven
  - Parameter values can be easily modified as they change in real world
  - Allow the user to customize their "look"
- Modal
  - Functionality is based on the Operational mode of the SAF Workstation since it is expected to be different
- Consistent
  - Based on the CCTT User Interface Style Guide
  - User learns one, can easily use another

G-70

Other Topics

- Development Risks
- CCTT Integration

G-71

SAF Development Risks

- CCTT SAF Requirements Push the State of the Art for SAF Systems
- Short Schedule Requiring Highly Productive Development
  - Effective Design Process
  - SW Process that Minimizes Testing Cycle
- Uncertain Delivery and Functional Completeness of ModSAF

G-72

The CCTT System Design Makes Use Of Common Software And Requirements To Reduce Cost And Increase Correlation

- CCTT has many components in common across the system
  - Terrain Database and Algorithms
  - Communications software
  - DRA and DIS interfaces
  - Damage Assessment
  - PVD
  - Ballistics
- CCTT has many common entity models across the system
  - OC and SAF
  - SAF and Manned Modules
- Use of Common Software, Designs, and Requirements by CCTT Ensures that:
  - Correlation of Data and Models is maximized
  - Duplicate Development Activities are Minimized

G-73

Several Approaches Are Used To Take Advantage Of Commonality Across The System

- Use of same models in OC and SAF common areas
  - Common models creates immediate correlation
  - Common models prevents duplication in costs
- Reuse of common system software
  - Algorithms correlate directly for such items as conversion between coordinate systems and dead reckoning algorithms
  - Prevents duplication of costs
- Reuse of common requirements at appropriate fidelity levels
  - Analyses are performed to match requirements at the SAF fidelity levels
  - Models are chosen or created which provide correlation with other systems
- Use of common models, data, and algorithms
  - Central source of CCTT models and data ensures correlation
  - All model and data usage is traced back to source of input for consistency

G-74

SUMMARY

CCTT SAF Meets Unique Challenges

- CCTT SAF Design and Development is Occurring in a Unique Framework
  - Active, Daily User Involvement
  - Use of Validated Models
  - Thorough Requirements Analysis and Documentation
- CCTT SAF is Based on Reeengineering ModSAF Design Concepts
  - Develop a New SAF System That Leverages Off of Lessons Learned by Industry
  - Developed with Broad View of CCTT and CATT Requirements
  - Developed with Life Cycle Maintenance as Major Factor
  - Developed to Support Research Reuse
- Horizontal and Vertical Scalability are Required to Directly Support CCTT as well as CATT SAF
- CCTT SAF Requirements Include an Unprecedented Level of Autonomy and Support for Operator
- CCTT SAF Capabilities Push the State of the Art in a Production System

G-76

The CCTT SAF Concurrent Engineering Team is Developing For the Full Software Life Cycle

The CCTT SAF System is:

- Maintainable: Life Cycle Considerations Have Always Driven Basic Architecture and Design, Resulting in Hybrid Approach Under 2167A
- Reliable: Traceability From Requirements, Formal Regression Testing of SAF Behavior, and Continual User Involvement
- Believable: User Involvement with SAF Development and Regression Testing, Explicit Tracing of Decision–Making Behavior
- Traceable: Audit Trail From Requirements (e.g. Doctrine) to Code
- Understandable: User Friendly UCI, Explicit Tracing of ARTEP–Based Rules in Militarily Relevant Terms
- Verified: CCTT SAF has Validated Models, User Involvement, and Testing Throughout Lifecycle, Giving the Fidelity and Confidence for Tactics Development and System Evaluation

G-77

CCTT SAF Openness and Traceability Provides the Basis for Future Research and Development

The CCTT SAF System Is:

- Extensible: Use of Flexible Architectures at Platform and Unit Levels, Connected by the SEOD Protocol, Supports Multi–Vendor, Multi–Echelon CGF
- Reliable: A Supported, Tested Platform on Which to Build
- Scalable
  - SEOD Allows Distributed Computing, Distributed C2
  - Generic Platform Interfaces and Tasks Provide Horizontal Scalability
  - Separate Inference Engine Provides Sophisticated Arbitration and Reasoning for Vertical Scalability
- Customizable: Offline CIS Editor, Edit CIS Parameters, UCI Customization
- Portable: POSIX, Standardized Ada, X / Motif, no Hardware Customizations

G-78

CCTT CGF Architecture Backup

- Overview
- Architecture Prototype
- Terrain Prototype

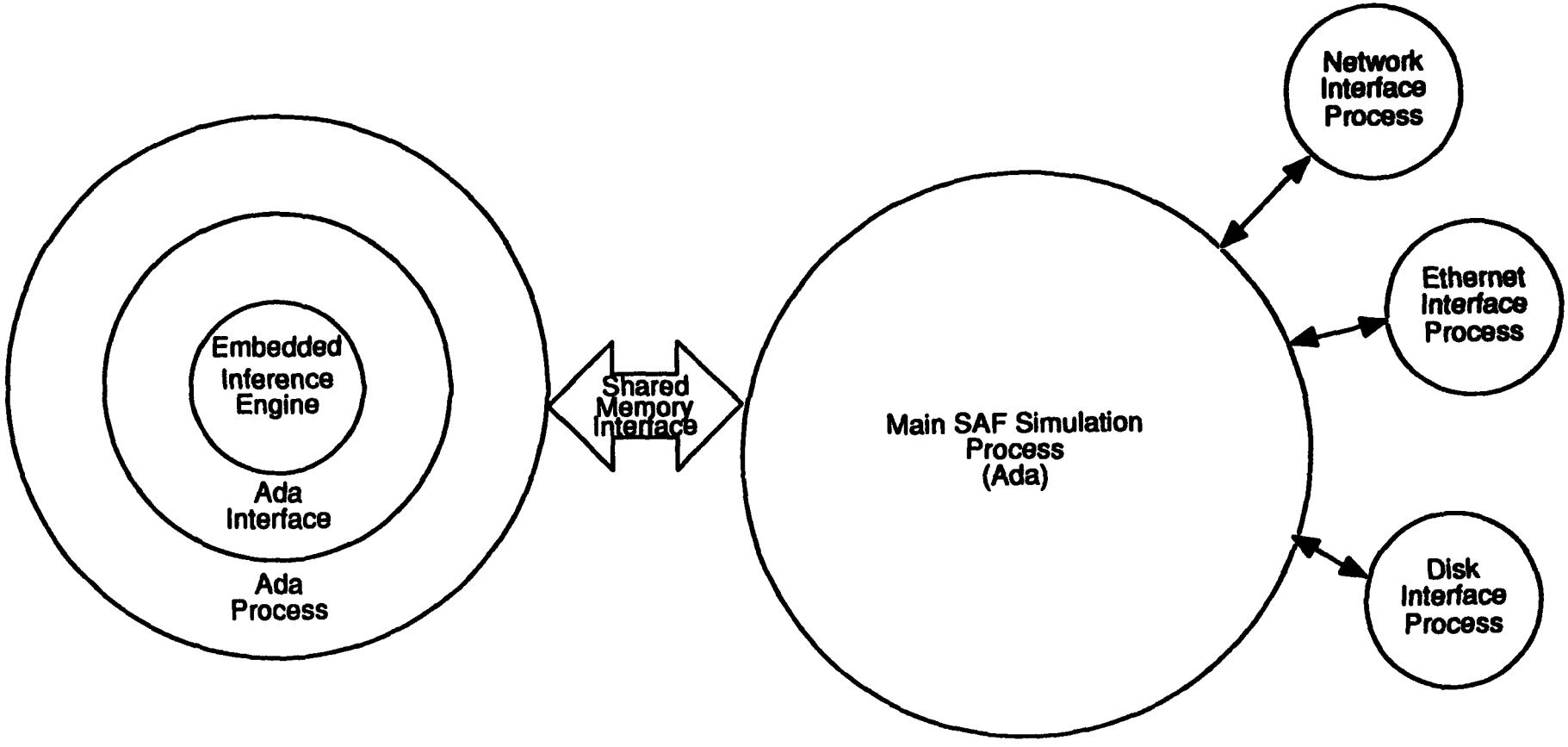
The CGF Design Maximizes Extensibility And Consistency

- The CGF file system provides for the central storage of CGF exercise files
  - Overlays
  - Exercise Descriptions
  - Terrain files
  - Task Organizations
  - Unit Descriptions
  - Rule Files
  - Initialization files
- Provides a central repository for all user workstations to access and share the same information
- These databases are easy to modify, upgrade, replace, and share

Provides openness, consistency, and ease of maintenance and extensibility.

G-80

The SAFsim Contains Multiple Processes To Support The Real–time Nature Of The Main Process In An Environment With Blocking And Slow Elements



This approach ensures an open approach allowing multiple vendor or organic solutions to tactical reasoning.

G-81

The SAFsim Contains Multiple Processes To Support The Real–time Nature Of The Main Process In An Environment With Blocking And Slow Elements

- Tactical reasoning is performed in the inference engine process
- The inference engine represents a slower process and can be executed at a slower rate without effecting the simulation
- The FDDI Network interface, Ethernet interface, and Disk interface represent service processes that buffer information which is consumed by the main process
- All communication during the simulation is performed by high speed memory interfaces directly supported by language constructs (no machine dependencies)
- The main process does not depend on the inference engine to finish for it to continue
- A blackboard paradigm is used to communicate between the inference engine process and the main process

This approach allows any method of tactical reasoning providing an open design for multiple vendor solutions or a home–grown solution

G-82

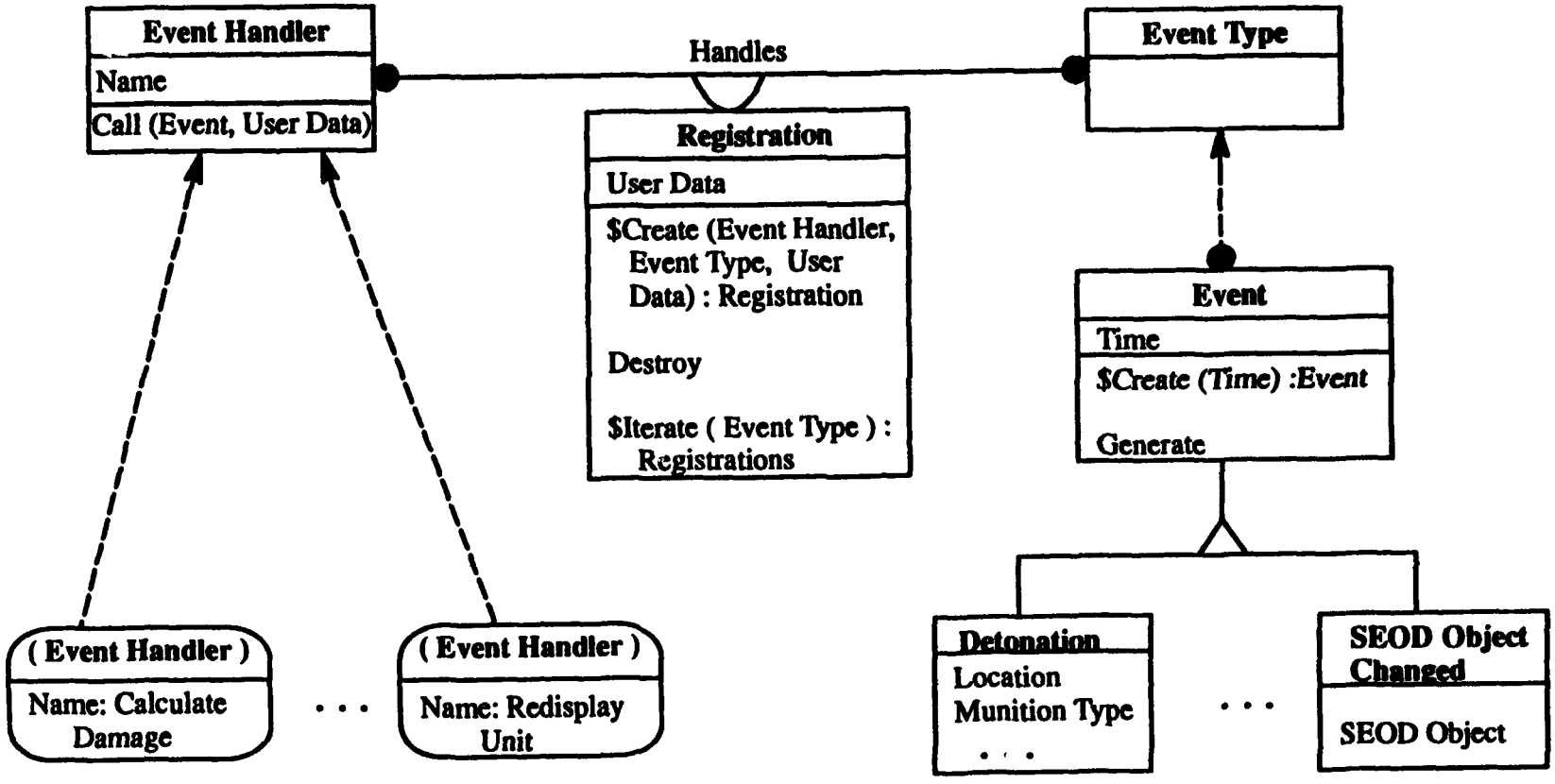
Architecture Prototype Backup

The Prototype Event Manager Distributes Events to the Appropriate Event Handlers

- Purpose
  - Decouples Event Generators (Primarily DIS Manager and SEOD Manager) from Event Handlers
  - Distributes Events Based on Event Type
  - Enables Mapping Between Event Types and Event Handlers to be Determined at Run Time
- Products
  - Language–Independent Design Expressed as Rumbaugh Object Model
  - Ada Package Specifications and Bodies
  - Example Program Produced by Direct Translation of ModSAF `LibCallback` Test Program

G-84

The Event Manager Object Model Presents a Language–Independent Static View of the Event Manager



Event Types Should be Defined at Compile Time, Regardless of Implementation Language

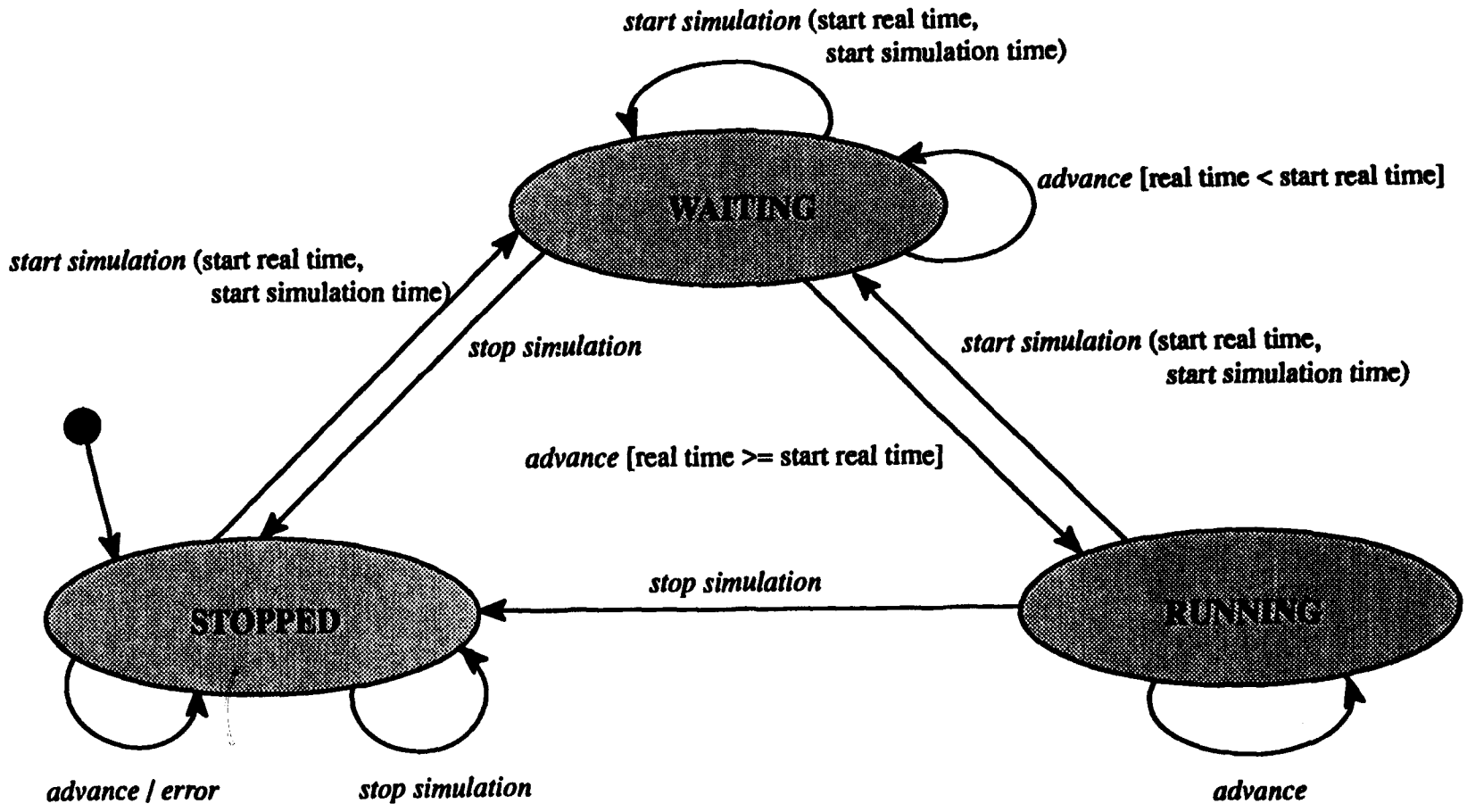
- Definition of Event Types
  - ModSAF Event Types are Defined at Run Time
    - Limits Types and Quantity of Parameters Passed to Event Handlers
    - Interprets Parameter Profiles at Run Time
  - CCTT SAF Event Types are Defined by a Variant Record in Package `Events`
    - Enables Compiler to Verify that Data Provided by Event Generator Conforms to Event Type
  - Recommendation: Define Event Types at Compile Time, Regardless of Language

G-86

Event Handlers Should be Designated Via Function Pointers, if the Implementation Language Permits

- Designation of Event Handlers
  - ModSAF Event Handlers are Designated via Function Pointers
    - Localizes Code Changes when Creating or Removing Event Handlers
  - CCTT SAF Event Handlers are Designated by an Enumerated Type in Package `Event_Handlers`
    - Enables Compiler to Verify that Event Handler has the Correct Profile
  - Recommendation
    - C: Use Function Pointers
    - Ada: Designate Event Handlers with Enumerated Type

The Dynamic Model of the Prototype Clock Package Includes Three States: Stopped, Waiting, and Running



G-88

Usage of Ada Package Calendar to Implement the Simulation Clock Trades Performance for Portability

- Portability
  - Resolution of POSIX `time()` Interface is 1 Second
  - Ada Package `calendar` Enables Compiler Vendors to Offer Arbitrary Resolution, Via Standard Interface
  - Result: Ada Implementation is More Portable than C Implementation
- Performance
  - C Implementation Enables Direct Control over Which C/Unix Library Functions are Called to Get Real Time
    - Function `gettimeofday()` is Preferred, Because it Returns Relative Real Time
  - IBM Ada Package `calendar` Calls Function `gmtime()`, Which Returns Absolute Real Time
  - Result: Ada Implementation is Slower than C Implementation

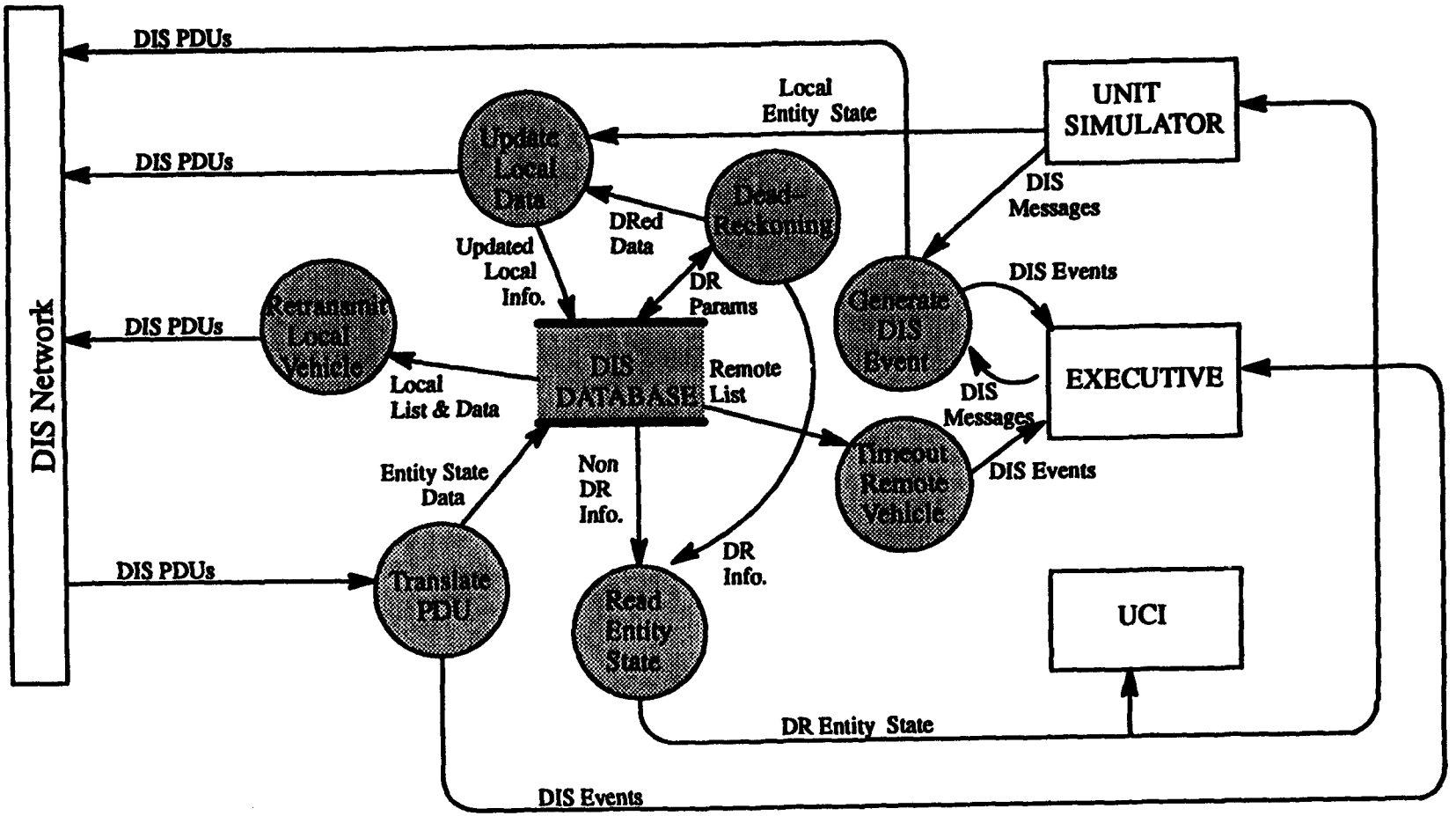
G-89

The Prototype DIS Manager Encapsulates the DIS–Related Requirements into a Single Reusable Component

- Purpose
  - Record Entity State Information for All DIS Entities (Local and Remote)
  - Perform Dead Reckoning per DIS Standard
  - Transmit Entity State PDU's for Local Entities per DIS Standard and CCTT System Architecture
  - Generate Appropriate Events for Incoming DIS PDU's
  - Provide Interfaces for Local Entities to Generate DIS Events
- Design
  - DIS Database is Fixed–Length Table
  - Table Index is Derived from DIS Entity ID Using Closed Hash Function
  - DIS Network Interface and Dead Reckoning are CCTT Reusable Components
  - DIS Manager may Become a CCTT Reusable Component

G-90

Data Flow in the Prototype DIS Manager is Centered on the DIS Database

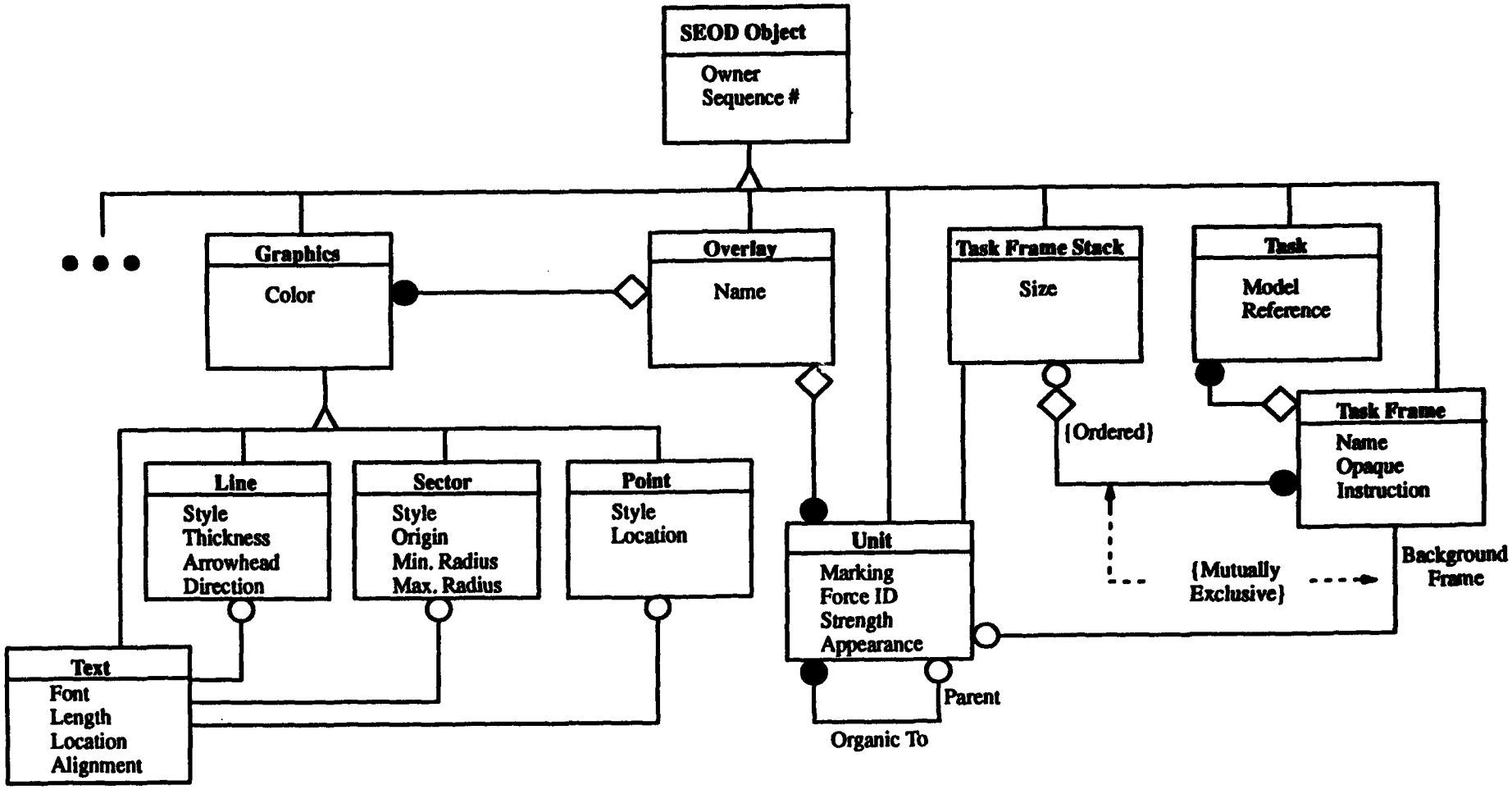


The SEOD Manager Enables the Distributed SAF Processors to Provide a Cohesive SAF Capability for CCTT

- Purpose
  - Enables Communication Between SAF Processors of Information Not Standardized by DIS
  - Supports:
    - Command and Control of SAF Units
    - Allocation of SAF Units to SAF Simulation Processors
- Design
  - Derived from ModSAF Persistent Object (PO) Database
  - Employs Rudimentary Object–Oriented Paradigm
    - Object Classes (One Level of Object Classes Specialized from `SEOD Object`)
    - Associations Between Objects (Unidirectional)

G-92

Object Modeling Concepts Are Especially Helpful in Documenting the Structure of the Shared Object Database



G-93

Several SEOD–Related Design Issues Remain to be Settled During Preliminary Design

- SEOD Object Model Will Evolve During Preliminary Design
  - Likely Areas of Change:
    - Overlays / Execution Matrices
    - Aggregate Units (Platoon, Company, etc)
- General Support for Bidirectional Associations May be Required
  - Depends Upon Complexity of Object Model
- Impact of SEOD Traffic on CCTT LAN Will be Estimated
  - Baseline Approach is to Use Separate Ethernet LAN for SEOD Traffic
  - Possible Enhancement:
    - Selectable Update Rates for SEOD Objects

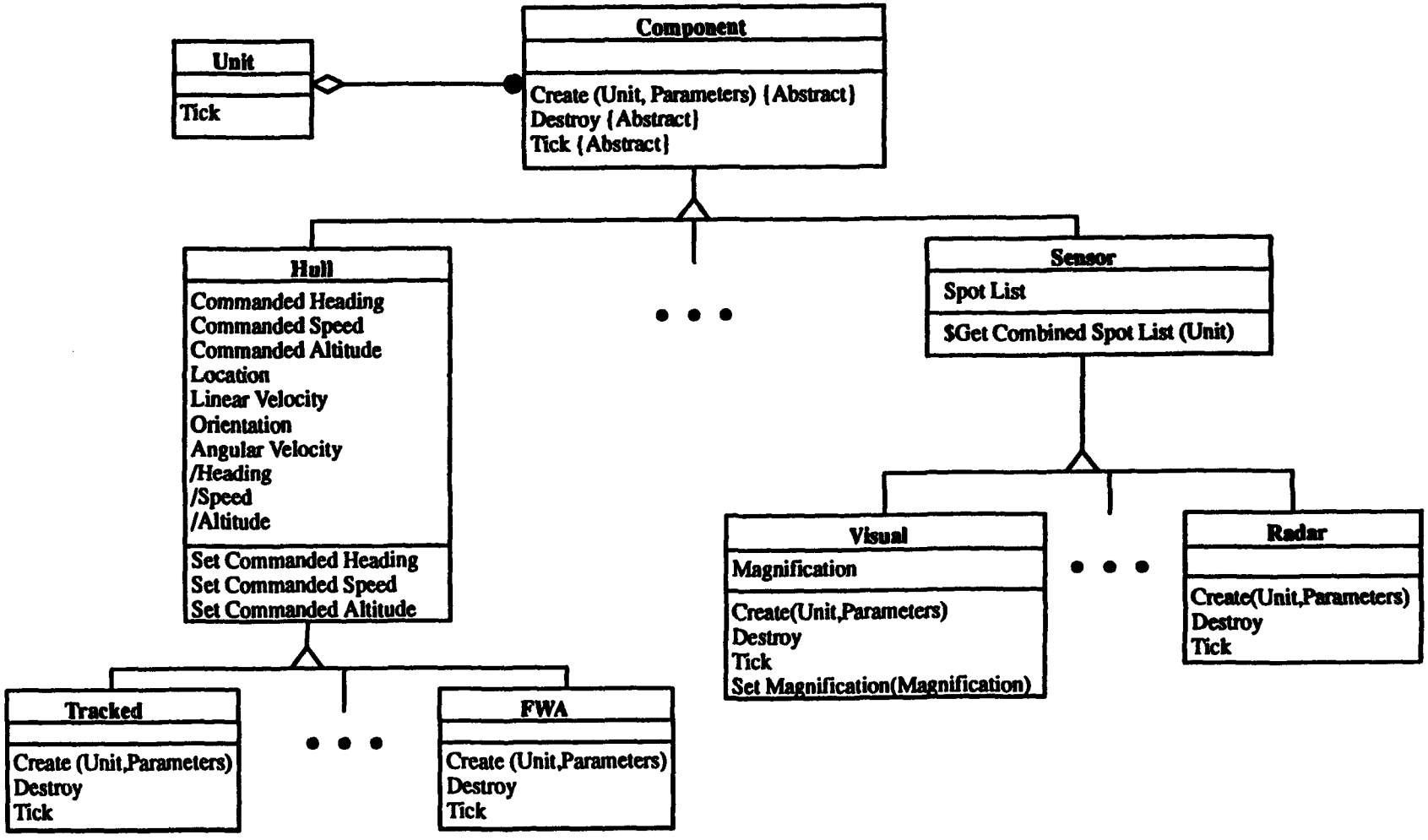
G-94

Generic Interfaces to Physical Models Provide a Powerful Method of Generalizing Tasks

- Purpose
  - Supports Dynamic Binding for Abstract Operations that Must be Provided by All Components
    - Create
    - Destroy
    - Tick
  - Provides Standard Interfaces to Operations and Attributes Within an Abstract Class of Components
    - Tasks Written to Generic Model Interfaces are Applicable to a Wide Range of Vehicle Types
- Design
  - Follows the Standard Object–Oriented Approach, as Shown on the Next Slide

G-95

The Generic Model Interfaces Follow a Traditional Object–Oriented Approach Based on Inheritance



G-96

The Prototype Ada Solution is Simple, Yet Provides All the Required Capabilities

- Dynamic Binding
  - Abstract Operations on Components are Implemented via Ada `case` Statement on Component Type
- Generic Model Interfaces
  - Separate Ada Packages Provide the Common Attributes and Operations for:
    - Hull
    - Turret
    - Gun
    - Sensor
  - Dynamic Binding is Also an Option at This Level
- This Work will Contribute to the General Object–Based Programming Approach for CCTT

G-97

Terrain Prototype Backup

G-99

Process Used for the Terrain Prototype To Identify Algorithms For CCTT SAF

- Perform a survey of existing algorithms
- Select algorithms for analysis
- Develop requirements for the terrain database
- Perform an analysis on performance of algorithms
  - C version of the algorithm
  - Ada translation of the algorithm
  - Ada optimized version of the algorithm
- Report the results

G-100

The Survey Identified Several Sources Of Algorithms Which Were Analyzed For Their Application To CCTT SAF

- The survey was based on several industry sources
  - Loral ADS
  - IST Studies
  - Military Operations Research Society's ORSA Handbook
- Studied the efficiency of the algorithms
- Studied the application to CCTT SAF requirements
- Studied the database requirements for size and content
- Ultimately settled on ModSAF algorithms using the CTDB and Quadtree database

Terrain Prototype Timing Analysis

- Overall Ada performance compared well to C
- Final metrics demonstrated that Ada performance was 1.08 times C with run–time checks disabled
- Code optimizations were minimal based on duration of prototype
- Algorithms can be retuned to an Ada representation without much effort

G-102