Spring 2000

Applications of agent architectures to decision support in distributed simulation and training systems

Plamen V. Petrov
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

APPLICATIONS OF AGENT ARCHITECTURES TO DECISION SUPPORT IN DISTRIBUTED SIMULATION AND TRAINING SYSTEMS

by
Plamen V. Petrov

This work develops the approach and presents the results of a new model for applying intelligent agents to complex distributed interactive simulation for command and control. In the framework of tactical command, control communications, computers and intelligence (C^4I), software agents provide a novel approach for efficient decision support and distributed interactive mission training. An agent-based architecture for decision support is designed, implemented and is applied in a distributed interactive simulation to significantly enhance the command and control training during simulated exercises. The architecture is based on monitoring, evaluation, and advice agents, which cooperate to provide alternatives to the decision-maker in a time and resource constrained environment. The architecture is implemented and tested within the context of an AWACS Weapons Director trainer tool.

The foundation of the work required a wide range of preliminary research topics to be covered, including real-time systems, resource allocation, agent-based computing, decision support systems, and distributed interactive simulations. The major contribution of our work is the construction of a multi-agent architecture and its application to an operational decision support system for command and control interactive simulation. The architectural design for the multi-agent system was drafted in the first stage of the work. In the next stage rules of engagement, objective and cost functions were determined in the AWACS (Airforce command and control) decision support domain. Finally, the multi-agent architecture was implemented and evaluated inside a distributed interactive simulation test-bed for AWACS WDs. The evaluation process combined individual and

team use of the decision support system to improve the performance results of WD trainees.

The decision support system is designed and implemented a distributed architecture for performance-oriented management of software agents. The approach provides new agent interaction protocols and utilizes agent performance monitoring and remote synchronization mechanisms. This multi-agent architecture enables direct and indirect agent communication as well as dynamic hierarchical agent coordination. Inter-agent communications use predefined interfaces, protocols, and open channels with specified ontology and semantics. Services can be requested and responses with results received over such communication modes. Both traditional (functional) parameters and non-functional (e.g. QoS, deadline, etc.) requirements and captured in service requests.

APPLICATIONS OF AGENT ARCHITECTURES TO DECISION SUPPORT IN DISTRIBUTED SIMULATION AND TRAINING SYSTEMS

by
Plamen V. Petrov

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer and Information Science

Department of Computer and Information Science

May 2000

APPROVALS

APPLICATIONS OF AGENT ARCHITECTURES TO DECISION SUPPORT IN DISTRIBUTED SIMULATION AND TRAINING SYSTEMS

Plamen V. Petrov

Dr. Alexander D. Stoyen, Dissertation Co-advisor — Date
Associate Professor, Computer and Information Science, NJIT

Dr. Gary Thomas, Dissertation Co-advisor — Date
Professor, Electrical and Computer Engineering, NJIT

Dr. D. C. Hung, Dissertation Committee Member — Date
Associate Professor, Computer and Information Science, NJIT

Dr. Franz Kurfess, Dissertation Committee Member — Date
Assistant Professor, Computer and Information Science, NJIT

Dr. Peter Ng, Dissertation Committee Member — Date
Professor, School of Computer Science, UNO

Dr. Ami Silberman, Dissertation Committee Member — Date
Assistant Professor, Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Plamen V. Petrov

Degree: Doctor of Philosophy in Computer and Information Science

Date: April 2000

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer and Information Science
  New Jersey Institute of Technology, Newark, NJ, 2000

- Bachelor of Science in Computer Science
  University of Central Florida, Orlando, FL, 1995

Major: Computer Science

Selected Presentations and Publications:

P. V. Petrov, A. D. Stoyen, "Compiler Support for Non-intrusive Monitoring and Debugging of Real-Time Systems in the CRL Environment," 1997 IEEE Real-Time Systems Symposium, San Francisco, California, USA, December 1997.

A. D. Stoyen, T. J. Marlowe, M. F. Younis, P. V. Petrov, "A Language Support Environment for Complex, Distributed Real-Time Applications," Proc. 3rd IEEE International Conference on Engineering of Complex Computer Systems, Milan, Italy, September 1997.

P. V. Petrov, A. D. Stoyen, "AWACS Weapons Director Trainer Tool: Architectural Design," Proc. 25th Workshop on Real-Time Programming, WRTP 2000, Palma de Mallorca, Spain, May 2000.

to

Daniela, Vasko, Roumiana

and

Ivi

ACKNOWLEDGEMENTS AND PERMISSIONS

The work presented in this dissertation has been sponsored in part by the Office of Naval Research, the National Science Foundation, the United States Air Force Research Laboratory, the Office of the Secretary of Defense (through the Naval Surface Warfare Center, Dahlgren Division) and other sponsors. Some of the early research was conducted in part at the New Jersey Institute of Technology and at 21st Century Systems, Inc (21CSI). Detailed additional research and the entire concrete software design and implementation have taken place at 21CSI. The evaluation has taken place partially at 21CSI and partially at various Air Force sites. 21st Century Systems, Inc. has generously granted permission to the author to present and publish the results of this work for the purpose of completing his dissertation research. All information in this report is unclassified.

The author would like to express sincere gratitude to his advisor, Dr. Alexander D. Stoyen, his committee members, 21st Century Systems, Inc, and the department of Computer and Information Science at NJIT. This work would not have been possible without the support and guidance from the colleagues and friends at these organizations: Dr. Samuel Schiflett and Dr. Scott Chaiken at Brooks AFB, Dr. Linda Elliott and Mathieu Dalrymple at Veridian, Dr. Philip Craiger and Gregory Myers at 21st Century Systems, Inc.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS

(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 The Importance of Agent Technologies for Decision Support

1.1.1 Description of the Problem Domain

In today's complex command and control management environments the time for planning and decision making has a significant impact on the effectiveness of the operation. Often, decision makers work under conditions of high stress, compressed time and overwhelming information flow. The criticality and the detail-prone processing of information to form an optimal decision make the task extremely difficult and time consuming. If the process of evaluating incoming information, developing and updating a strategy, distributing and monitoring its execution by the operational team (in real-time) and evaluating the results could be automated by software tools, then the effectiveness of the team of decision makers and of the mission as a whole will increase considerably. This iterative process is in the heart of modern strategic and tactical planning. The purpose of this work is to model automation, analysis, and tradeoff software agents that will significantly reduce the turnaround time of the process and will thus enhance mission effectiveness. Additionally, these intelligent agents must operate in real-time, as new sensor and munitions information arrives, to dynamically evaluate the changing tactical picture and to adjust the mission parameters accordingly, in order to achieve optimal results under the specific circumstances. It will be foolhardy to believe that such agents, no matter how sophisticated, could ever replaces human decision makers. Our goal is to

aid the decision makers in time consuming, mundane processes and to offer analyses of their decision and alternative recommendations, in order to help those decision makers operate in a less stressful, more focused environment.

The problem domain we have selected is specifically related to military operations – we are focusing on command and control decision support for AWACS teams of operators. It is important to note, however, that the problem is quite common among a diverse set of domains, which include decision makers, both military and civilian. Domains range from the obvious civilian applications to air traffic control, to business planning and resource allocation, to financial market planning to emergency and rescue services etc. Thus the provided solutions may have a very wide applicability to both defense and commercial areas.

In this work we study the properties and tradeoffs associated with software agent environments for complex distributed decision support systems. In a modern software simulation for command and control training, various models of distributed decision support are strongly desirable, but on the market, very few (practically none) products exist which could even claim such features. Even in the research community, agent-based design and implementation of decision support systems are rarely discussed. We see a large gap between the need for sophisticated tools and integrated environments for decision support on one hand, and the existing tools, which merely aim at enabling team interaction or simulation, on the other. Implementing decision support tools, without a thorough analysis of the ramifications of the functional and non-functional requirements

of such tools, could be counterproductive, quite expensive and even dangerous for large command and control systems. The majority of decision support solutions, which have emerged so far, have not incorporated performance-based tradeoffs, and have relied predominantly on techniques of the mid-to-late Eighties and early Nineties. These techniques have emphasized cognitive task analysis, information presentation, and human performance evaluation. While we agree that these features are important for effective decision support tools, we approach the problem by analysis of performance concerns and tradeoffs, such as speed versus safety and efficiency across heterogeneous environments. These issues have not been addressed, primarily because there has been little incentive to focus on them in the known solutions (which emphasized interoperability and interconnectivity, as opposed to efficient performance).

1.1.2 Developing Technologies to Address the Problem

With the development and wide availability of large-scale, heterogeneous networked computing, the landscape of software technologies is quickly re-focusing to support this new paradigm. Courtesy the proliferation of LANs, client-server architectures, the Internet and the World Wide Web, fields like agents-based computing are transitioning from the software research labs to real-world applications. The demand for mobile components, which can migrate, adapt and perform at wide varieties of hosts, is growing very quickly. Mobile applications will soon become typical and common on high performance, distributed, heterogeneous, parallel and large-scale platforms. Consequently, the development of systems that utilize performance-oriented agent-based,

potentially mobile components, is of major importance. These systems need to consider the management of tradeoffs between efficiency, safety, predictability of time and space use and other concerns. Significantly, a well-developed mobile agent-based environment will free up the programmer (and the user, and the system administrator) from having to consider performance and other tradeoffs manually or in ad hoc manner. Thus, all pertinent aspects affecting resource allocation and use in the system, especially in the software system layers (system and networking software, middleware, language support and services, application libraries and support), need to be taken into account systematically and through means that will provide the programmer and the user with greater flexibility, while allowing him to focus on functionality rather than on component management.

Mobile code. Distributed computing, historically, attempts to solve the problem of limited performance of stand-alone supercomputers by assigning tasks to multiple, less expensive (and, of course, less powerful) computers working in parallel, and combining their results in a meaningful way. Mobile code may be considered as one of the underlying technologies for distributed computing, which enables programs to be sent from one computing node to another. In contrast, in the message passing model for distributed computing data is passed (wrapped in messages) between stationary processes; in the client-server model requests are passed from the clients to the server (in some cases requests may include algorithmic descriptions), and the server sends back a reply, after processing the request. Mobile code can co-exist with message-passing or

with the client-server model, providing additional/alternative functionality. In fact, typically, mobile code is implemented on top of a message-passing environment.

In the recent years there has been quite a bit of activity and discussion on the so-called network-centered computing concept and its enabling technologies, one of which is mobile code. The paradigm of mobile-agent computing brings profound advantages to distributed computing, at a price, which seems acceptable for today's network-enabled computing resources. Mobile agents, which travel to bulky data, communicate and execute in open, heterogeneous environments, present a promising alternative to the traditional client-server approach. Agent-based applications represent more naturally the interactions in many problem domains by shifting the focus from implementation details to reducing overall complexity. Mobile code overcomes many limitations of traditional approaches, caused by immobility of bulky or sensitive data. Thus, mobile code techniques are steadily gaining popularity. The stage of hype is slowly fading and now we are facing the serious problem of providing mobile-agent environments, robust enough to support the design and development of complex systems.

Software (Intelligent) Agents. This is a fairly loosely used term, which has not been and perhaps cannot be defined precisely. In most cases intelligent agents are intuitively defined as "pieces of software able to function (and to move around in a network) autonomously and to interact with each other and with users." In this work, we prefer to simplify our definition and view intelligent agents as software components which have a specific task to perform, can communicate with human users and/or other agents and may

be capable of moving from one computing node to another, either following a human command or some internal algorithm or script. We will carefully avoid the term "intelligent agents", since this work is not focused on AI or its applications, *per sè*, and thus we would feel it is out of scope to try to define "intelligent". We view software agents form a systems architecture and design perspective, as building blocks for a new computing paradigm, which provides the desired flexibility, extensibility, and interoperability with legacy systems, to build an efficient real-time decision support system for interactive simulations. Agent technology has matured to the point that it is ready to be applied to realistic problems with strict requirements and constraints.

We have evaluated and studied the approaches to multi-agent systems taken by a number of research and industry leaders. Many routes have lead to disappointments or unsatisfactory results; other efforts, while very promising, are still in their infancy stages. We have identified a few projects that have contributed significantly to the area with well-designed, realistic and implementable (or already operational) models. However, we have not been able to pinpoint models to address the rigorous needs of high performance decision support simulation and training, most likely due to its highly domain-specific requirements. Thus, we have chosen to develop our own agent-based environment, building upon knowledge gained from successful designs and avoiding the mistakes in the failed prototypes. We need to build our system on top of an environment, which simultaneously provides a reasonable set of primitives, is widely accessible, and is sufficiently flexible and programmable to allow us independent implementation of features if necessary. One prominent example of a well-balanced language and

environment is Java™[1], which aims at reasonable safety, accessibility and object-oriented designs, coupled with a large set of external libraries.

Multi-agent systems with military applications. Improved military operational capabilities already require and will continue to necessitate constant progress in the technologies for information management systems for the C^4ISR (computer-based command, control, communications, intelligence, reconnaissance, and surveillance) applications. One area of technologies, in which significant work is expected is the area of *human-agent systems* (HAS). Agent, by definition are computational entities (software programs) that assist human operators in accomplishing a particular task. Agents can be imbued with the ability to seek out information over distributed networks; search through information databases in varying formats (including structured, unstructured, hierarchical, relational [H96]); manipulate information through filtering, transforming, aggregation, and fusing of multiple, independent information steams [HSK98]; and to report information to the human requester [T99]. Typically, there may be a number of agents working on several tasks at any point of time, e.g., several agents monitoring and filtering information from disparate channels, agents to aggregate and fuse relevant information, agents to select an appropriate visualization (text vs. 2-D tables vs. 3-D animations: cf. [CMS99]) of the data to report, and so on. Some agents may exhibit a high level of autonomy, allowing them to make critical decisions based on information found without human intervention or guidance. This may be advisable only

[1] *Java is a trademark of Sun Microsystems, Inc. We will omit the trademark symbol from here onwards.*

when the response time required for a human decision would greatly reduce the decision quality.

The problem remains as to how best to capture, filter, aggregate, fuse, and display the right information at the right time. Moreover, to increase the chances of military success and maintain an acceptable level of survivability, these systems must exhibit: a) real-time performance, b) event-driven behavior, c) distributed decision making and reasoning under uncertainty, d) the ability to learn, e) information fusion, f) target track management and deconfliction, and g) reconfiguration to different platform mixes. Below we provide a scenario, which demonstrates these capabilities in the context of a large, hybrid, multi-agent system implementing control technology in a combat situation. Later, we will describe some of our work, which demonstrates how current technologies supports each of these attributes.

Multi-objective optimization. The problem of decision support can be viewed as one of optimizing a multi-objective allocation of resources. In a generic mission, typically there are a number of tasks to be accomplished, which determine the mission success. Each task can possibly be accomplished by a number of means – either involving different resources, or different timing or both. The training systems should be able to evaluate decisions about resource-to-task pairings made by human commanders (or trainees) and to suggest optimal or near-optimal allocations, all in the relative time-frame allocated for the decision. The system's goal is to assist in making recommendations to the human decision maker, or implement automated decisions, which allocate the resources, subject

to the conflicting objectives, constraints, in the best possible way under significant competition for the resources. Given the general growth of the problem space the process has to consider, the synthesis and cost and objective engines must make use of heuristics and approximate "best effort" strategies (of course, *exact* allocation is computationally prohibitive). A set of heuristic approaches has to be applied in order to provide useful resource allocations under the existing time constraints. These may include well studied algorithms as well as new, domain-specific methods – genetic algorithms, artificial neural net, projection, fuzzy logic, probabilistic, greedy, and so forth.

Intelligent, heuristic searches. These will be based on our work in heuristic representation, search, conflicting objective representation, and integration (scaling, fusing, converting) of individual cost functions into overall objective functions. Essentially, the information and the problem space in the domains of interest are inherently exceptionally large and complex. Exact analysis to support decision making in the AWACS world is NP-hard (even if it is in fact computable at all). Thus, the problem space needs to be reduced, and the decision-making analysis in turn needs to be facilitated and undertaken in a computationally feasible (though heuristics) manner.

Utility, forecasting and economics of uncertainty theories. The decision-making process is often very challenging due to the presence of incomplete, confusing and partially correct (and partially incorrect!) information. Thus, to model friends, foes and the environment, and to provide functional, timely and otherwise relevant advice to command and control we cannot rely on precise or even statistically-averaged

information models. Instead, we have to make use of theories suitable for modeling information and structuring decisions in the presence of incomplete, partially correct information and under conditions of time and mission stress. In general, utility theory recognizes that humans do not, in most cases, decide on the sole basis of probabilistically established information. Rather, they rely on a mixture of judgement calls, experience, risk aversion and other subjective factors, in addition to basic probabilities. Economics of uncertainty brings in additional factors into the process, specifically to compensate for uncertain information. Forecasting under uncertainty compensates further to hedge risks against predictions based on both unknown (inductive) and incorrect assumptions. In this work we have considered decision-aids under conditions of uncertainty, though our decision support models do not yet fully incorporate unknown or imprecise information.

System and software architecture. Our implementation is based on an open-ended distributed object-oriented software architecture. An extensive class hierarchy designed to allow for (1) high-level environmental and combatant entities (various types of air-borne, land- and sea-based vehicles and weapons, terrain, weather, etc.), (2) detailed structures to model the behavior of relevant entities with sufficient detail (planes, tanks, rocket and missile launchers, etc.), (3) intelligent autonomous entities and human players (decision aid agents, commanders, human-substitute agents, and so on), and (4) services (distributed communication, information management functions, object and agent management, computationally-intensive library of functions and so on). The architecture defines interactions between the entity simulation services and the agent management environment as the core of the application. We employ agents to interact with and advise

human participants, additional agents to coordinate among agents and humans, agents to play roles of various objects in the simulation, and so on.

Simulation and applications. The applications partially developed in this work can ultimately be used for three related but distinct purposes. Initially, we envision that the software will likely be used by a relatively small control and evaluation group of experts, in order to verify and validate DSS effectiveness in complex command and control domains. Once validated, the system may be extended, based on feedback from the evaluation, before being used on a regular basis in a domain-specific training and operations. In the training application, interactive simulations are widely used to improve the skills of personnel without the risks and costs of training on real systems. However decision support has not been applied with the same success to simulated training of decision makers. We argue that, since DSS will be integral part of all future command and control support systems, the training exercises will greatly benefit from integrated simulation with decision support.

1.1.3 Our Approach: Performance-aware Multi-agent Architecture for Decision Support

We will now outline the specific contributions of this work. While the field of preliminary research for the concepts presented here was relatively wide, including real-time systems, resource allocation, agent-based computing, decision support systems, and distributed interactive simulations, our major contribution is the development of a multi-agent architecture and its application to decision support for military simulations and

command and control training. An architectural design for the multi-agent system, featuring a number of functional classes of agents, their interactions and dependencies were outlined in the first stage of the work. Next, the rules and objectives, including objective and cost functions were derived for the decision support in the AWACS (Airforce command and control) domain. Finally, the architecture was implemented and evaluated on a command and control simulator and trainer for AWACS WDs. Part of the evaluation process took place at 21st Century Systems, Inc., while another, independent evaluation was undertaken by the Human Effectiveness team at Brooks AFB, including current and retired WDs, and other training personnel.

Distributed multi-agent architecture. We developed a distributed architecture for performance-based management of multiple autonomous interactive agents. The approach is significantly different than previous efforts and augments the state of the art with new agent interaction protocols, agent performance monitoring and remote synchronization mechanisms. The multi-agent architecture supports peer-to-peer inter-agent communication as well as hierarchical coordination, where hierarchies can be defined on the fly. Communication takes places via static interfaces, dynamic protocols, or over open channels with a pre-defined ontology and semantics. Communications are most often requests for services, responses with results, or notifications for monitored data. Service requests are controlled by functional (traditional) parameters as well as non-functional (e.g. QoS, deadline, etc.) requirements.

Decision support agents and their interactions. Our distributed multi-agent architecture was applied to the team decision support problem domain. Rules, objective and cost functions, as well as new heuristics were developed to form a network of agents, which autonomously perform situational and risk assessment, resource allocation and fitness evaluation, and, based on rules and heuristics, recommend a set of actions to the commanders. The contribution in this phase is integrated design of heuristic resource allocation and multi-agent system and its application to highly performance sensitive interactive simulation and decision support. To our knowledge, even though independently resource allocation and agent systems are well understood, their combined utilization in performance-critical real-time applications, such as decision support systems, has not been explored so far.

Implementation and evaluation. We implemented the multi-agent architecture within the AWACS WD Trainer Testbed (courtesy 21st Century Systems, Inc. and the USAF). The implementation was entirely in Java and featured agents, agent scripts and interaction protocols, situational awareness and threat assessment heuristics, resource allocation rules and heuristics, and performance measures. The prototype was tested internally at 21CSI as well as by current, retired and in-training WDs at Brooks AFB. To support flexibility and extensibility, while still maintaining strict control over performance, the multi-agent architecture was built and integrated with the simulation engine in-house. A number of minor and moderate engineering challenges had to be overcome in order to provide an effective implementation within the time and budget constraints of the project. (a thirty-month small business innovation research (SBIR)

project funded by USAF Research Labs). Specifically, due to the ongoing maturing of the Java language, the effects of performance glitches needed to be accounted for. Some of these were alleviated by the introduction of Just-In-Time (JIT) compilers and, recently, full binary compilers for Java. Similarly, Java's remote method invocation (RMI) mechanism, while relatively straight forward to use, has evolved significantly from its initial stages. Additionally, the support for platform-independent graphical interfaces leaped forward when Java Swing (a.k.a. Java Foundation Classes – JCF) was introduced. All of these factors, combined with the inherent Java performance difficulties, presented a non-trivial implementation problem.

Testing and validation of the prototype was possibly the most rewarding phase of the project. The agent-based decision support was evaluated at two stages, independent of each other. First, an extensive internal evaluation and testing was conducted, based on scenarios and guidance from experts from the Airforce. Second, an independent group at Brooks AF Base applied the prototype in training exercises with novice and experienced WDs, who evaluated the DSS and the human-agent interactions. Though we received very positive feedback from the field evaluation at Brooks AFB, in this work we will mainly focus on the in-house evaluation. The experiment was aimed to examine the change in performance of WDs with or without the help of the agent-based DSS. Initially, the interface with agents and the usage patterns were studied. Once enough data was collected to conclusively indicate that the users were in effect following agent recommendations (rather than observe and ignore them), a set test scenarios were executed with and without the agent support. The collected performance data shows that

the agent recommendations did not decrease the performance and in some cases dramatically improved the success rate of the simulated WDs.

1.2 Overview of the Dissertation

Let us now take a brief tour of the balance of this dissertation. The work is presented in seven chapters including this introduction, in which we have described the problem domain and have outlined our approach. In Chapter 2 we describe the selected problem in more detail. We describe the use of intelligent agents for command and control decision support, as we address the specific requirements posed by the problem domain. Then, in Section 2 of that chapter we elaborate on the factors connecting agent-based decision support and distributed interactive simulations, an approach which is a major contribution of this work. Section 3 in Chapter 2 looks at the challenges of the selected problem from both computation-theoretical and software engineering points of view.

Chapter 3 dives into the details of our approach. In Section 1 we present a distributed multi-agent architecture. In Section 2 we present an agent architecture for distributed interactive simulation – detailed conceptual, functional and system level descriptions are provided. Section 3 focuses on the management and synchronization of the cooperating agents in a time- and resource-constrained environment. Section 4 in chapter 3 focuses on the performance requirements and the timing constraints for the multi-agent system.

Chapter 4 presents prior and related work we have examined and built upon. We focus on two relevant fields – agent architectures, in Section 1 and resource allocation, in section 2. We present a comparative description of other agent architectures in Table 1.

In Chapter 5 we provide detailed discussion of the experimental prototype. Section 1 provides a global view of the agent-based decision support system and simulation test-bed. In Section 2 we discuss all aspects of the design and implementation. Issues such as communications and physical entity models, command and control model, fault modeling and tracking, visualization and so on are considered. Finally, in Section 3 the agent management and interaction model are discussed.

Following is Chapter 6, which presents the evaluation methods and results. Section 1 in this Chapter focuses on the measures and methods used to evaluate the effectiveness of the agent-based decision support system. Outlined are two types of evaluation – direct evaluation by problem domain experts, and indirect, through the performance improvement of trainees, using the system. Section 2 presents the results and findings of the experimental evaluation.

We provide a conclusion in Chapter 7. First we summarize the results of the work in Section 1, as we emphasize on the key contributions of this dissertation. Next, we outline directions for future work associated with agent-based decision support in distributed interactive simulations. At the end of the dissertation a list of publications can be found.

CHAPTER 2

PROBLEM DEFINITION

2.1 Intelligent Agents Applied to Decision Support

2.1.1 Command and Control Domain Specific Issues

Current simulation and training environments do not support adequately team training for command, control and decision making, especially through specialized agents and visual interfaces. Specific shortcomings include lack of support for: (1) user hierarchy and multiple views for use by different participants at different stages of the mission or exercise; (2) non-traditional but important objectives such as maximizing performance, maintaining secure, reliable, timely operation in the presence of uncertain information and faults, graceful degradation under faults (and other Quality-of-Service measures), and for open system design that enables the use of generic tools. The agent-based environment presented here overcomes many of these shortcomings by assisting command and control team members in making more informed and accurate decisions in less time and under less stress.

Extremely large problem space. The monitoring, control and direction of multiple entities to engage in a coordinated mission requires the acquisition, processing, evaluation and display of enormous amounts of information. The sheer number of entities

involved, with their individual properties, capabilities and objectives produces a significant stream of information. When mission outcomes need to be evaluated based on a particular resource-to-target allocation and on the current tactical situation, the potential for an exponential explosion of the number of states of the system prevents an exhaustive search. Similarly, resource allocation is an exponential problem, which, considering the number of entities cannot be solved optimally under any realistic timing constraints.

Real-time performance requirements. One of the requirements for command and control decision support systems is real-time or near-real-time performance. In dynamically changing environments, such as an active theater of warfare, the timeliness of a decision is often as important, if not more important that the type of the decision itself for the successful outcome of the operation. Similarly to the complete futility of a weather forecast which arrives a day late, a late decision is not only worthless but could be potentially harmful. While different military units deal with widely varying response time requirements and deadlines, we must assume that the shortest deadlines must be met to ensure the synchronous operation and interaction of interdependent units. Thus we base our system requirements on the command and control of fighter aircraft – one of the fastest machines ever built, with most demanding control systems among the military arsenal.

In addition to the domain-specific timing constraints, we may encounter implementation dependent requirements and constraints. Due to the fact that all entities are simulated, we must conform to the simulation update rates to be able to use current information and to

keep the user updated with timely decision aids, displayed as the view of the simulated world is rendered. In future extensions of the work, it is conceivable that some of the entities will be introduced as live sensor feeds, such as radar tracks, sonar information etc., while some will still be simulated (for training exercises. This model will necessitate new synchronization and timing constraints, which need to be supported by the developed architecture.

Potentially high-risk, high-payoff of a critical solution. The application of agent-based decision support to the command and control domain presents challenges which are best described as high-risk high-payoff. This fields is extremely dependent on information technologies, and even slight improvements in the quality of data, presentation models or processing algorithms can have high impact on the decision quality of commanders. On the other hand, psychologically, expert decision-makers are less likely to trust new technology more than their strongly developed intuition. Thus, as requirements for command and control decision aids, the system should have similar look-and-feel to what decision makers are used to, while providing qualitatively enhanced information and options that may have been overlooked due to overload.

2.1.2 Decision Support with Multiple Cooperating Agents

Decision support tools have traditionally focused on the decision-making process (from a cognitive psychology, or operations science point of view). In this work we approach the problem as one of information flows, resource allocation and heuristic evaluation of

alternatives. Framed in this way, the goal of aiding a decision-maker lends itself nicely to a naturally distributed solution, such as applying software agents. Essentially, we develop an multi-agent architecture to model the information pathways in command and control decision making, as well as to construct and evaluate alternative decisions. The work is defined within the context of distributed simulated AWACS WD training, though the same principles should apply to any tactical decision making process.

Information monitoring, filtering, transformations and presentation. A significant subset of the agents are dealing with information flows in the system. This includes Monitoring Agents, which are responsible for timely updates on tactical information, such as location and movement of hostile and friendly resources. Monitoring Agents interface directly with the simulated state of the world, or with sensor processors to capture and store available data. Although Monitoring Agents may perform limited filtering (e.g., to avoid redundant information), a special class of agents – Filter Agents – performs the bulk of the selective distribution of incoming information. Filter Agents maintain subscriber lists for data items needed by other agents or human users. If data is not available in timely manner (as expected), exception messages are propagated from the Monitoring Agents to Filter Agents and further to data consumers, to help them adapt to the degraded data streams. Further in the pipeline, Transforming and Fusing Agents modify, correlate and prioritize data according to the needs of the agent community, determined ultimately by specifications (implicit or explicit) from human users.

Strategic analysis and evaluation of the situation. To achieve and maintain situational awareness, a class of agents is tasked with analysis and evaluation of the tactical picture. The SA Agents evaluate data obtained by the Monitor and Filter Agents and structure it to gain understanding of current tactical environment and the trends, changes and differences from the expected state of the universe. Situational awareness traditionally is defined to mean not only knowledge about the current state of affairs, but also understanding of how it differs from what the expected state was. Thus, situational awareness is a fundamental component of any decision support system. SA Agents perform searches, comparative evaluations, maintain a database of beliefs (of how the universe should look) and determine the differences between the incoming information and those beliefs. Important events are identified and prioritized by their significance to the decision-makers, indicated by objective functions.

Generation of alternatives and heuristic evaluation of decisions. Once the hot spots in the tactical picture are identified by the SA Agents, a different class of agents takes over the pipelined information. The Evaluator Agents are capable of constructing potential decisions or allocations, based on domain-specific rules, and evaluating those allocations upon the current set of objectives. Alternative decisions may be formed as simple platform-to-target pairings, or as more sophisticated group formation and mission planning process (e.g. a Strike group is preceded by a SEAD group). Optimal or near-optimal resource allocation can be extremely computationally intensive, and thus the Evaluator Agents are equipped with a set of heuristics to help cut down the allocation space and arrive at a relatively good allocation solution relatively quickly (at the expense

of loosing guaranteed optimality). The process of constructing alternative allocations is naturally complemented by the evaluation of the alternative's fitness to the overall mission objectives. The Evaluator Agents then select a subset of the alternatives and present them to the decision-makers as recommendations via the User Interface agents (discussed next). Another application of the Evaluator Agents allows the system to monitor and assess decision-makers' performance. By letting some Evaluator Agents run in the background and compare the human's decisions to what other Evaluator Agents recommend as near-optimal decisions, the tool can either provide critique to or learn from human decision-makers, depending on the current mode.

User interactions, customization and adaptation. A special subclass of agents, UI Agents, are responsible for interacting with the human users. While some other classes may produce information for the users, and others may require user input, the UI Agents sole purpose is to facilitate a better communication with the user. Utilized techniques include customization per user, adaptive changes to the user behavior, interfaces with advanced visualization (3D, VR, multi-media) and others. Some UI agents also serve as online tutors, help agents, guides and wizards.

2.1.3 Management of Agents with QoS and Real-time Constraints

In our view, quality of service and real-time requirements are part, to variable extent, of all modern large software systems. This is especially true for decision support systems, where quality and timeliness of information have crucial effect on the quality of

decisions. The need for QoS and real-time performance is further emphasized by the specific application of DSS to military command and control simulation and training, where the decision-makers are faced with a rapidly changing environment, uncertain information and high stakes (potentially life-or-death) on decisions. Accordingly, the software system that supports those decision-makers should adapt to such a hostile environment and provide QoS and RT performance in a dependable manner. The challenge in our work is twofold: 1) derive reasonable QoS and RT constraints for C^2 DSS applications, and 2) design and implement a multi-agent DSS that satisfy these constraints. While it is quite common in many areas to impose such constraints, multi-agents systems have not traditionally focused on strict QoS enforcement and strong RT performance. We claim these constraints are not only essential for the problem domain, but also can be satisfied by a well-designed multi-agent system.

Performance monitoring and evaluation. Performance, as part of the non-functional characteristics of software components (along with fault tolerance, security, etc.) can be described by functions and tables associated with each execution code segment, as derived by compilers, profilers and other tools. We will not focus on determining the performance characteristics of software components, but rather on the issues of utilization of those descriptions and on the run-time enforcement of performance and QoS promises. This approach makes certain assumptions, based on the fact that we develop a multi-agent, potentially distributed system. We assume that a multi-processing operating system is available, which supports light-weight processes (e.g. threads) to implement the agents' multiprocessing backbone. We further assume reasonable control over a fair

scheduling mechanism, as well as reasonable access to performance data gathered by the operating system. The raw performance data available to the multi-agent manager (from various OS sources) is then collected and associated with the QoS requirements of agents. Since the agent manager is aware of the non-functional specifications of agents, it is capable of correlating actual performance data with performance requirements (and specifications).

QoS checking and enforcement. To ensure performance criteria are met, specific QoS requirements need to be enforced. While performance monitoring provides information on whether or not non-functional objectives are being met, this information needs to initiate a feedback loop to control the QoS of each agent. If the agent manager detects degradation in performance, it could take one or more of the following actions: 1) notify the agents whose performance is degrading, so that they can start using more efficient algorithms, requiring less resources, 2) modify the resources allocated to a subset of the operating agents, such as CPU time (by changing the scheduling priority), available memory, etc.; 3) move to a different node or suspend agents to improve the load balance and/or modify the network traffic. Changes in the internal behavior of agents, such as selecting a less computationally expensive algorithm, are delegated to individual agents, since problem-specific information has to be taken into account when selecting those alternative computations. If the agent is not capable of adapting its behavior to meet the QoS requirements, the agent manager may be able to replace it with another, cheaper, less precise agent, though replacing an active agent together with its context may be an

expensive operation. Thus a tradeoff exists between retrofitting agents with potential long-term benefits and the immediate mitigation of degraded performance.

Safety and fault tolerance. We recognize that there exist multiple tradeoffs between different quality of service measures and requirements. For example, an emphasis on more security will require additional resources to support the security algorithms and thus will likely increase the execution cost of an agent. Similarly, fault tolerance seems to be in conflict with minimizing the number of agents operational at any one time and thus will increase the overall load in the system. In order to achieve a balanced set of QoS constraints that can be guaranteed to the user, the system has to maintain a dynamic equilibrium by considering all the existing tradeoffs among individual QoS requirements. This can be achieved by the systematic utilization of the non-functional descriptors of each agent. Essentially the agent manager on each node monitors and maintains a balance among the agents and their QoS requirements. Agent managers on different nodes need to communicate if an agent is about to be moved (by its own request or initiated by the system) to ensure that the proper resources exists on the target node to support the mobile agent's QoS requirements.

2.2 Agent-based Decision Support in Distributed Interactive Simulations

While being relatively young, the field of distributed interactive simulation has evolved and matured notably. Interactive simulations today pose significant requirements on any tools that claim to interoperability and integration with such simulators. Further, today's

simulation technology enables realistic modeling of a large amount of entities with rich behaviors, which presents a realistic and very challenging environment for command and control decision support. In this section we address the issues of dealing with significant information flows and solving a large exponential problem, which are inherent in the application of DSS to distributed interactive simulation.

2.2.1 Simulation Data Filtering, Processing and Navigation

The issue of information management becomes critical with the realization of the sheer amount and rates of data flows, involved in realistic military simulations. Only a fraction of this data stream, however, is relevant to any single decision, and the challenge is to efficiently identify these data and to capture, categorize and keep updated only what is significant. Therefore, it is important to maintain situational awareness on an ongoing basis, in order to be able to select data that is essential, as the situation dynamically changes. In fact, information monitoring and selective updates are no longer a one-way process, delivering data form the sources to their consumers. A fair amount of continuous feedback to the data monitors has to take place, to ensure that appropriate filters are always focusing on the current top-priority information. These needs can be accomplished by a multi-agent software architecture, which is flexible enough to adapt to the changing environment and robust enough to deal with large amounts of data in an efficient way.

We identify two distinct challenges in this area. First, the design of a conceptual model of an agent team, which will gather, filter and organize data is of interest. This model must allow for a flexible specification of objectives and efficient, robust mechanism for satisfying those objectives. The specifications need to include a prioritization scheme for what data is considered most valuable, as well as a cost function for the collection of these data. Second, the integration of the agent community with a distributed event-driven simulation presents a significant engineering challenge. The agents should be allowed efficient access to the simulation data, without the risk of corrupting it or reading an inconsistent state. Feedback mechanisms should be provided from the agent team to the simulation, to model virtual data collection platforms (such as reconnaissance aircraft for example). Despite the apparent need for a tight integration, the model should allow for a modular design, so that both the simulation and the agent components could be independently substituted with other systems.

To reduce the amount of data examined at each simulation update, a variety of filters can be applied to the search. First, *geographic constraints* could be placed on the entities of interest. This works well in the cases when an area of influence could be defined in the context of current tactical operations. Second, a constraint on the *types of entities* could be used to weed out irrelevant information. Third, a *relational subset* could be determined – this will involve the selection of entities related in some pre-defined way to the set of entities of interest. For example, we could be monitoring a Strike group in Quadrant 4, and we'd like to determine if there are any hostile aircraft threatening its mission. The geographic filter will limit us only to entities in Quadrant 4. The type filter

will eliminate all resources that are not aircraft. The relational filter will select only those aircraft in Quadrant 4, which are armed and the Strike group is in their weapons ranges (the latter conditions could be any relation the user describes). Though the use of filters could reduce significantly the amount of data considered by the DSS, they should be applied with caution, not to lose any important information. Specifically, a feedback mechanism needs to control the application of filters, based on the needs and requirements of the DSS and ultimately of the decision-makers.

2.2.2 Decision Support as Resource Allocation

Essentially, we view the process of decision support as general resource allocation (a decision is in general about pairing tasks or targets with resources capable of handling the tasks) with multi-objective satisfaction constraints, functional and non-functional properties. The system's goal is to recommend to the human decision maker, or implement automated decisions, to allocate resources, subject to the conflicting objectives and constraints, in the best possible way (there is possibly significant competition for the resources). This allocation process differs in level of detail only, across different stages of the decision making process. Losses or failures to complete the mission result from poor allocation choices. Given the exponential growth of the problem space, the process has to consider heuristics and approximate (best effort) strategies must be used – exact allocation is computationally prohibitive. We provide a range of heuristics, which can be called explicitly by the user, or by the system under general instructions given by the user. These include commonly available and "home-grown":

genetic algorithms, neural nets, simulated annealing, projection, greedy algorithms, and other methods for heuristic traversal and optimization of large problem spaces. Three modi operandi are supported in the model: search (entire space is considered per iteration), construction (a fixed number of objects are considered per iteration), and hybrid (the heuristic relies on a strategy/user to switch modes; e.g. first, do a quick global search, then optimize-construct a local region...).

A core problem to decision support is reacting to dynamic changes in the system. Humans and computational support resources, each in their own way, need to make quick assessments and adjustments to maintain situational awareness in dynamic environments. The basic need is to react to such changes incrementally, rather than starting from scratch. We have investigated aspects of incremental computation and how it applies in different areas. We have constructed resource allocation heuristics which assess quickly future real-time performance of a workload of objects and tasks by looking at the allocation so far and by projecting the effects of the remaining assignment [SMCG96, SHH87, SBAM96]. We extensive experience in incremental computation for dataflow analysis, which is a key technology for failure and performance analysis, component selection and management, simulation, and other activities, and we have defined a framework for incremental verification and of complex systems [Metalii93, MR90, FMY92, MMR95, SMHY93, SM92, YMS94, YTMS95, AMS96].

The decision support tool suite makes use of utility, cost, and objective functions, which capture requirements and performance capabilities of resources, objectives and the

overall battlespace. Individual functions are fused, scaled and converted into a suitable, integrated objective function, which is in turn used to measure the fitness of a resource or an allocation. We have researched such functions extensively. In the overwhelming majority of reported efforts [CP91, H90, LMA88, H91, L88, S77], cost functions are represented as constants or scalar variables. The objective function is consequently computed as a linear expected-value summation, with either constant or constant-sum weights, representing significance of each cost function's contribution. However, such approaches may fail to represent adequately the situation, for a number of reasons, including the following: (1) significance values may change over time, (2) individual objectives may exhibit a dependency on each other, (3) the integrated objective relationship may not be linear. Consequently, and on the basis of our extensive experience in tools and languages [S87-SG92, MSMW94, KS86, YMS94, YTMS93, LMS96, AMS96, YSG91, HS91, HMSS95], we have developed both a detailed hierarchy (with multiple inheritance, e.g. a mission cost function derives attributes from both real-time weapons delivery and time-over-target objectives) and a general equational form for an objective function, derivable recursively on the basis of the hierarchy [DRSL98]. In brief, a function is represented recursively as a function of more detailed cost functions, divergence, fusion, scaling functions, and various algebraic combinators. Default definitions of these functions correspond well to simple, common objectives. Numerical method solvers are easily applied to this form (and thus, the form can be and is used in our tools).

As expected, the equational form supports conversion, scaling and fusion of cost function information. While conversion and fusion are typically found in discussion of cost functions, scaling is also very important. A classic example of use of scaling is in a HPCC system where the network is much slower than the general-purpose computers. Consequently, to avoid an integrated objective function which is very sensitive to network cost changes and very insensitive to computation cost changes, we may wish to express network timing as slower units (e.g., in 10^{-2} sec) and computation timing in faster units (e.g., in 10^{-4} sec). This approach also has some concrete fundamental characteristics vis-à-vis practical, non-preemptive properties [ABMS98].

2.2.3 Agent-based Approach to Decision Support

We take a pragmatic approach to intelligent agents, as we apply our experience developing agent technology to other domains, such as software systems synthesis, decision support under uncertainty and others. We view agents as autonomously running software, which monitors information and events, analyzes, and advises human users. Such agents add great utility to the users and developers of distributed decision support training systems. Agents provide choices and evaluate the performance of users "on the fly" to give valuable feedback both the trainers and the trainees. Agents also help the content developers in composing and fine-tuning exercise materials and test cases, relevant to the current task priorities. The trainers benefit from agents that search and evaluate the "fitness" of exercises and tests, while the content developer is working on scenario development.

We have developed three classes of agents in our tool suite. In the first class are human-role-playing agents. This type of agent implements human functions, such as team-members or enemy in an interactive exercise. Such agents are extremely useful in a distributed command and control training, when a collection of functions must be executed precisely, at a specific time and without being partial to a subset of the trainees. A human-role-playing agent is programmed through a script, which in turn consists of a sequence of orders. Authorized human users can alter the script at any time, even while the session is executing.

The second class is comprised of human-assistant agents. A human assistant agent provides concrete advice to humans who are participating in the session. Such assistant agents could be useful both in the training process as well as in the content development and evaluation processes. For example, if a trainee is making an incorrect choice at a critical point in the exercise on which the correctness of the rest is dependent, an agent can provide a second chance or additional clues for the concrete situation, thus highly increasing the value of the exercise for the trainee. Human assistant agents both react to human requests and provide advice without being prompted for it, if, according to their scripts, they determine that help and guidance is needed. Human assistant agents and the algorithms behind them are par of the contributions of this work.

The third class of agents is populated by coordinators and arbitrators of distributed decisions. Recall that multiple humans and multiple agents are partaking in parallel in a

training session. Even when individual agent and human decisions are based on team-level, global knowledge (which may or may not be the case, depending on the nature of a decision or knowledge available to a human or an agent), it is normal to expect conflicts and also temporal discrepancies. Agents of the third kind try alleviating such conflicts, advising other agents and humans, including, agents of the second kind, how to adjust these decisions and re-enter a more stable situation, while still not sacrificing their (possibly conflicting) objectives. Examples of conflicts include multiple users requesting modification of a single simulation object, such as issuing new orders, at the same time, or a user attempting to switch the nature or the pace of an exercise while in a team mode with other participants. Agents of the third kind too are a major contribution of this work.

2.3 Why Is the Problem Hard

The problem of decision support in the command and control domain is not new. One could argue that, when faced with the same problem ancient emperors used their favorite oracle as the decision support system of those times. Indeed, through the history, mankind has sought "decision support" in different forms and from different sources: elders, wise man, prophets and oracles, religion, gods, science, technology, and finally from computers. Oddly enough, in the theory of computing, the types of formal automata capable of solving problems equivalent in difficulty to the decision support problem, are indeed called oracles. We intuitively claim that, if stated and defined properly, decision evaluation and decision support in the context of military command and control are NP-hard, equivalent in difficulty to the hardest computational problems.

In addition to the challenging computational problem at hand, the enticing idea of applying software agents to the decision support problem presents a host of engineering problems. Specifically, one of the most interesting challenges is the integration of a multi-agent architecture with a distributed interactive simulation engine.

2.3.1 The Computational Problem

To reason about the complexity of a decision support system, one must define and constrain the problem. We will make a series of conservative assumptions, each simplifying the problem. We will then argue that the original decision support problem is at least as complex as the constrained version.

General setup. Let us consider a simplified model of our simulated world (which in turn is a model of the real world) – a model containing a set of "friendly" assets and "hostile" assets. Each asset possesses properties and capabilities, which determine its value and its threat to others. The intent of each side is to eliminate as many assets of the opponent as possible. This model is an extreme oversimplification of the original problem, because it eliminates: a) neutral forces, b) environment elements (terrain, weather, infrastructure, etc.), c) uncertain information, d) dynamically changing intent and rules of engagement. Thus the state of the modeled universe can be described at any given point in time by the two sets of assets and their properties. Properties include geographic position, current and maximum speeds, current heading, range, available munitions (with their capabilities),

and a set of objectives (targets). Again, this is a simplification of the original problem, where many other factors may be included in asset's properties and capabilities. In this context, a decision will represent a targeting pairing, such as "friendly asset X will target hostile asset Y". This is the simplest decision possible in our system – other, more complex decisions may involve grouping of friendly assets to perform a mission, using decoys and evasive maneuvers etc. – we will not consider any of those for the purpose of complexity evaluation.

Now we are ready to state two problems, which are part of the original decision support problem. We will normally argue that each of these sub-problems, in the constrained context, tends to be non-polynomial, and thus the decision support problem is also NPc.

Optimization. Given a set of decisions over a state of the modeled universe, select a subset of decisions, which provides the greatest benefit toward the objective. In other words, if any other subset of decisions were selected, the objective of eliminating the opponent's assets would have been achieved slower, with more losses, or perhaps not achieved at all. The problem statement implies that any decision can be described with a value, corresponding to its contribution to the objective. Deriving these values is the core of the problem. The value of a decision, however, cannot be derived in isolation of other decisions. It would be a fallacy to consider only the capabilities and projected positions of the platform-target pair. Not only do we have to consider the current targeting assignments (due to past decisions), but also we have to take into account the interaction of the decisions in the evaluated subset. Thus, we have a to perform selection of subsets

at least at three levels (set of current decisions, set of prior decisions, and set of assets), which forces the problem into non-polynomial solutions.

Construction. Given a state of the modeled universe, construct the most valuable set of decisions for this particular state. It is clear that this problem can be solved by first deriving all possible decisions and then applying the optimization problem to that set. Constructing all possible decisions for a given state is an exponential operation with respect of the size of the state. As we saw above, the optimization problem is also exponential. Applying this method, we are faced with an avalanche of one exponential problem feeding another. Unfortunately, no alternative solution has been found (which would, incidentally, prove that P=NP).

Another intuitive argument about the complexity of the decision support problem could be derived from a parallel with game theory. Any board game (checkers, battle-ship, chess) is in essence, solving a well-defined, highly constrained decision support problem. Let us consider the game of chess – an extensively studied, well understood, ancient game, challenged by computers since there were computers. Chess, originally intended as a "military command and control" problem, is in fact a very well defined, constrained and simplified decision problem. We do not need to point out how much a realistic (and still simplified) model of modern warfare differs from chess, and yet, all attempts to provide software that plays chess "well" have resulted in extremely costly solutions.

With these arguments in mind, we see that the decision support problem, even in its simplest form, is exponential in terms of the available data items (entities, properties etc.). Therefore, to provide an exhaustive solution would be, mildly stated, impractical for any reasonable size of the model. Thus we focus on alternative approaches, such as greedy algorithms, optimization algorithms, heuristic resource allocation and objective function evaluation.

2.3.2 The Engineering Problem

The successful design and implementation of the agent-based decision support tool depends both on developing the appropriate computational model as well as on designing a robust and efficient software architecture. The latter involves the resolving a number of software engineering problems, two of which are of particular interest. First, a multi-agent architecture needs to be developed, that supports QoS control and is relatively efficient. Second, the agent architecture needs to be tightly integrated with an interactive simulation, while still preserving a modular design.

Multi-agent architecture. Agent architectures for a wide variety of applications have been the center of attention for agent-based computing in the past years. While the filed offers significant progress in a number of areas, most agent architectures still have significant overhead, and worse yet, their resource needs and performance parameters cannot be easily predicted or even bounded. Driven by the needs of the decision support

application, we needed to focus our work to architectures that are efficient, predictable, while still remain reasonably flexible and open.

Integration of agents and simulation. Integration in general has emerged as one of the most challenging problems in software engineering. In our specific case, the need of integration of two computing paradigms – agent-based computing and interactive simulation – presented an interesting problem. The goal of the design was, on one hand, to keep the modularity of both the simulation engine and the agent-based environment, so that reuse and extensibility can be achieved, while on the other, efficient, timely data exchange was desired between the agents and the simulation. The approach we took features an agent manager, which is tightly integrated with the simulation engine, while individual agents are flexible, autonomous, and modular.

CHAPTER 3

DETAILS OF OUR APPROACH

3.1 Multi-agent Architecture

Agent-based systems are groups of agents that work together as a single system to integrate their functionality. They consist of a group or groups of agents that interoperate via a support infrastructure, cooperating to execute large tasks in a distributed manner. The individual agents are encapsulated, semi-autonomous processes that execute on a computer network, offering their services to other agents, other processes or users. Each agent is a specialist in a particular task or subtask. To execute a larger, more complex task, an agent-based system composes a solution to the task from the different services offered by the individual agents in its system. An important piece in this picture is the necessity for the agents to communicate among themselves to coordinate the execution of these complex tasks.

An agent-based decision support system can provide flexible and scalable environment for enhancing the capabilities of humans in command and control of large, complex military battlespace. The support and automation of such C4I problems poses a significant engineering challenge, beyond the capabilities of conventional tools and environments. First, the battlespace commonly involves multi-source information gathering, filtering and fusing (for example, airborne radar combined with satellite and ground-based surveillance), and a variety of domain-specific tools, such as simulators,

physical and mechanical models, information analysis tools, etc. Second, large command and control problems, such as a regional military conflict, tend to be unique, in the sense that it is highly unlikely that a similar event has occurred or will possibly occur in the future, because of the dynamics in the technologies, forces and political environment.

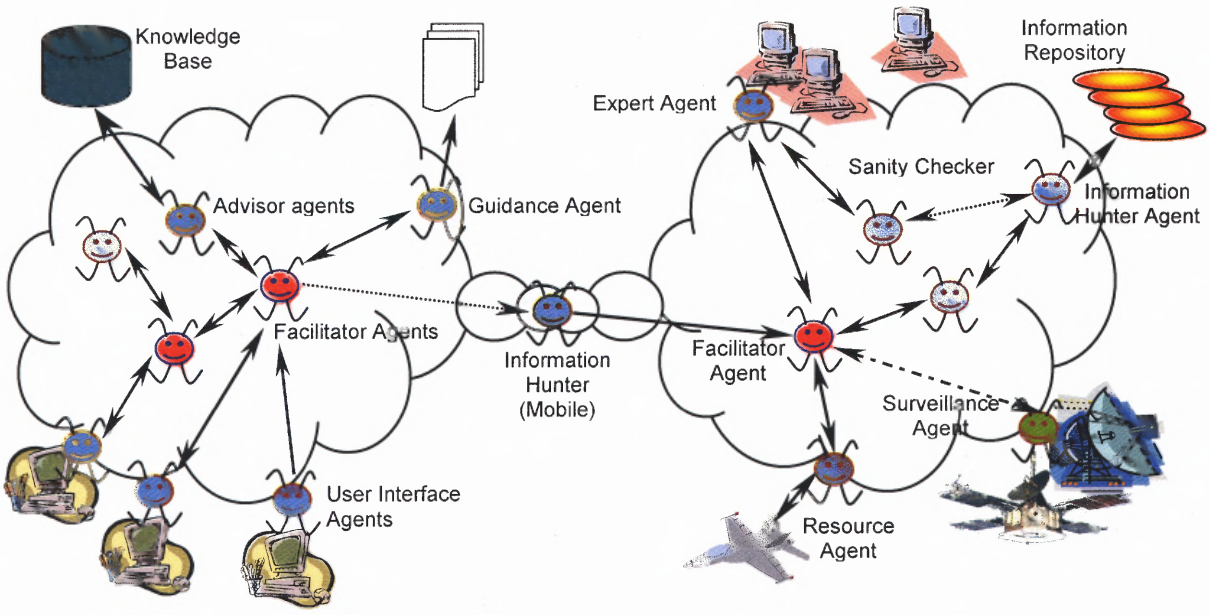

Figure 1. Multi-agent environment

This, combined with the enormous cost for military training, drives the development of cost-effective command and control simulation and training out of reach for traditional approaches. In contrast, highly adaptive communities of intelligent agents, each bringing a specific expertise, can be tailored very closely to the needs of a unique exercise. Third, autonomous agents not only can perform evaluations and suggest alternative and improved allocations during a simulated exercise, even without direct supervision of a trainer, but also, they can be deployed as part of an operational system to facilitate decision support, failure prediction and even self-testing.

The backbone of the agent-based environment is the distributed agent management medium. This component provides the basic "habitat" for agents; it facilitates agent interactions and ensures transparent distributed operations. The medium is populated with agent families, some of which capable of relocating from one computing node to another, others attached to specific resources.

Section 3.2 Agents in a Distributed Simulation Environment

3.2.1 Functional Description

The AWACS trainer tool is built around a distributed agent-based architecture. The server hosts a Simulation engine and an Agent manager – the heart of the system. The two parts share entity data and communicate via common memory space. The Simulation engine keeps track of all entities (creation, position, movement, capabilities, failures and ultimately, destruction) by referencing the appropriate physical and logical models. The Agent manager enables coherent interoperation of all agents. Individual agents can view the current state of the simulated universe through the Agent manager, which uses the exported interfaces of the Simulation engine.

Figure 2 shows the Client and Server layers of the distributed architecture and their interactions via Simulation data links and Agent interactions. The Agent Manager acts as a facilitator for agent interactions, as well as an interface to the simulation data, provided

by the Simulation Engine. The agents residing on the server side – Analysis, Navigation and Monitor agents – communicate directly with the Agent Manager, while the agents on the client side communicate to the Agent Manager via a network layer (Java RMI). For efficiency purposes, the Agents residing on the client side are allowed to communicate with each other (bypassing the remote Agent Manager) and to interact with the User Interface directly.

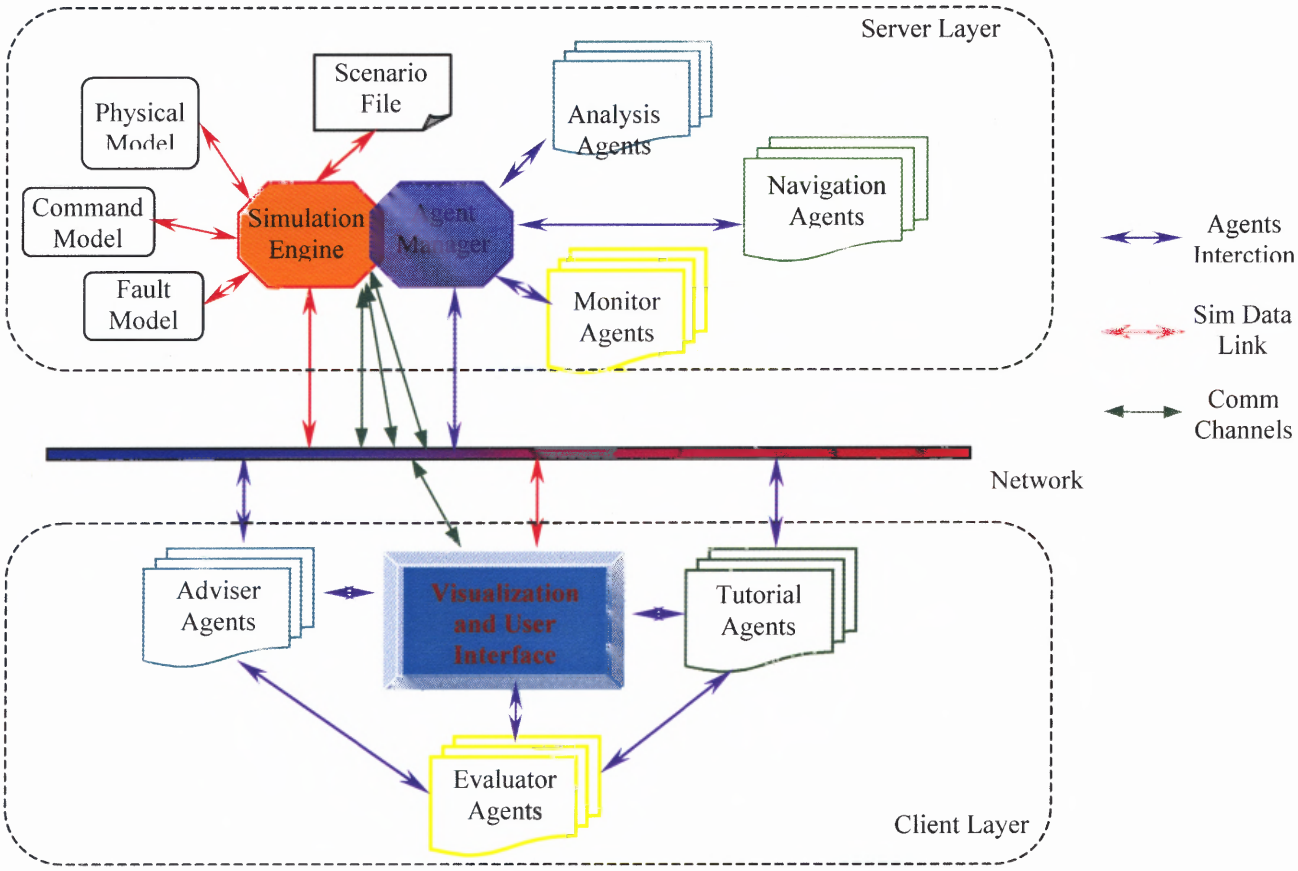

Figure 2. Distributed agent-based simulation and DSS tool architecture

The Simulation Engine is initialized from a scenario file with the number of entities, their type, position, and other properties, as well as with the number and type of roles expected to be played by humans. The scenario file can be referenced periodically for creation of

new entities throughout the simulation. The simulation engine then invokes the main simulation loop, which updates all entities according to their physical models, provides a heartbeat for the agents, and broadcasts updates to the clients. Simulation entities (or resources) behave according to their current orders, and their command, fault and physical models. If entities are not controlled by a human operator (such as a Weapons Director), the may follow a predefined script (from the scenario file) or will be guided by specialized Navigation Agents.

Agents in the server layer are organized in three cooperating groups: Analysis agents, Monitoring agents, and Navigation agents. The each of three classes may consist of multiple (possibly heterogeneous) agents. The function of Monitoring agents is perhaps easiest to understand – they are in charge of tracking various data items and providing feedback if certain anomalies are detected or thresholds exceeded. Data items of interest include platform properties (location, armament status, fuel status, damage), pilot/operator status (fatigue, SA, general abilities), mission status (primary & secondary targets, ROE), and others. An example of a Monitoring agent-generated event could be an advisory, such as "Attention: Bingo Fuel in 5 minutes" or a warning – "Warning: primary mission cannot be accomplished with the current armament of ..." or even a recommendation: "Recommend reverting to secondary target or RTB".

Navigation agents have a dual role. In the absence of a human operator, navigation agents are responsible for "driving the platform" – they simulate a live pilot or the logic behind a UAV's navigational system. In the presence of a human operator the Navigation agents

are still active in the background, evaluating the current route and providing feedback to the human. Possible tasks are obstacle recognition/collision avoidance, threat-along-the-route estimation, shorter/safer alternative path evaluation, etc. Example of events generated by Navigation agents are advisories – "Attention: shorter path to target found", warnings – "Warning: AAA zone ahead!", and recommendations: "Recommend break left 60° to avoid SAM site lock." In practice, navigational recommendations usually require immediate attention, and thus, to simplify the task of a WD we assume that such recommendations will be automatically accepted by the pilot of the affected platform.

Analysis agents perform the most computationally intensive task of evaluation of the current strategic picture, suggesting re-assignments and re-evaluating to find an improvement in the expected results. The basic strategy of this type of agents is to work in the background on alternative allocation of friendly resources to enemy targets and threats and to evaluate those alternatives according to a set of objective functions. The objective functions are typically set for each scenario, as they may vary depending on the specific missions. We also allow online modification of the objective functions (by a super-user) to increase the flexibility of the model and to ease scenario modification.

Analysis agents may apply variety of techniques to arrive at alternative allocations of resources. One approach is to start with a locally (for each WD) optimal greedy allocation and then to evaluate the global fitness of that allocation, iterating through the process similarly to simulated annealing. Another method, based on genetic algorithms is to incrementally improve on the current allocation, based on small perturbations in the

assignments. Similarly, other approaches can be taken, and, what is more interesting, multiple approaches can be competing against each other. Ultimately, if a threshold of "optimality" is exceeded, the Analysis agents generate recommendation events, such as "Recommend TARGET 2F15C-A to 2Mig23-X" or a set of similar recommendations. The human user is then responsible for accepting or rejecting these re-targeting recommendations.

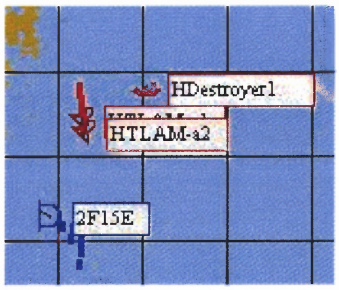

Figure 3. A pair of blue fighters (2F15E), originally targeting the red cruise missiles (HTLAMs), is now navigating around the hostile Destroyer's anti-air radar envelop.

While the agents within each class perform very similar functions, the interactions between agents of different types can produce quite interesting results. Just like in a real team, where ideally the value of the team effort is greater than the sum of the individual efforts, cooperating agents can provide greater benefit than each standalone agent can individually. Consider the following scenario. A pair of fighters is vectored to intercept two cruise missiles due North. Before the F15s are in range to engage the cruise missiles, they are illuminated by enemy anti-air radar on a Destroyer class ship. At this point the Monitoring agent detects and evaluates the threat posed by the possible anti-air assets of the enemy destroyer. It communicates a high threat event to the Navigation agent, which recommends immediate change of course to remain outside of the destroyer's radar envelop, and so the F15s turn North-West. This evasive action, however, is in conflict with the primary mission (target the cruise missiles) and the Navigation agent attempts to

return to the intercept course. The resulting path is a gradual tracing of the radar envelop (Figure 3). After a few minutes the Monitoring agent observes that the pair of fighters are low on fuel and recommends in-air refueling at the nearest Airforce tanker cell (Figure 4). At this time, the Analysis agent is triggered by the threat of the cruise missiles, which now remain untargeted. The re-assessment of the tactical situation shows that two pairs of Navy F14s are now available to target the hostile cruise missiles. New targeting recommendations are issued to that effect by the Analysis agent.

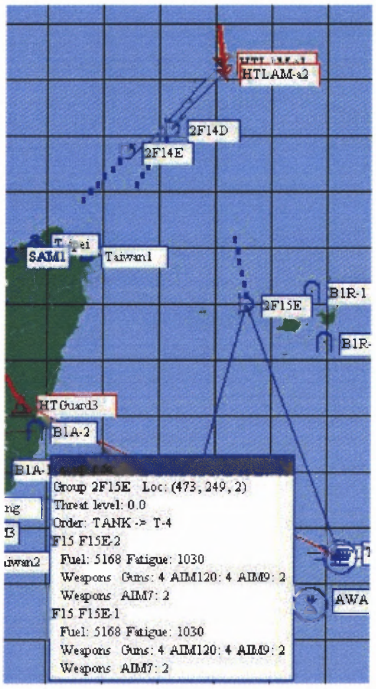

Figure 4. While the original pair of fighters (2F15E) targeting the cruise missiles has returned for refueling, two new pairs of fighters (2F14D and 2F14E) are now assigned to complete the mission.

This example illustrates the cooperative interaction between all three types of server-side agents. The cooperation is facilitated directly by the Agent Manager. The client-side agents, however can only access the Agent Manager through the distributed object interface on the network. Thus, the Agent Manager is sparingly used in cases where information needs to be synchronized or updated either on the server or the client side.

One such case involves transfer of resources from one WD to another. The process unfolds as follows. Initially an Analysis agent evaluates an alternative allocation, which gives us a tactical advantage. A set of re-targeting recommendations, which results from the new allocation, is forwarded to the Adviser agents at the client side (possibly involving multiple WD and multiple client-side agents). After a brief interaction with the user through the VUI, the Adviser agents report back to the server with information on accepted and rejected recommendations (from human users). In the meanwhile, the user-agent interaction is monitored and aided by the Tutorial agents, which provide context-sensitive help, when needed.

3.2.2 Conceptual Architecture View

In this section we will describe the conceptual design of the AWACS training tool environment. Based on domain expert and on experience with legacy systems, the goals we have set forth are as follows:

- Provide a training and operational DSS environment for teams of Weapons Directors
- Provide advanced visualization features, while preserving the look-and-feel of the familiar legacy system
- Be able to support physically-distributed team training
- Ensure the application is scalable for a large number of entities and arbitrary size and composition of teams

Figure 5 shows a conceptual diagram of the major architectural blocks of the System and their interaction. The core of the system is represented by the interaction between the Simulator and the Decision Support components. Entity data and Timing control are provided by the simulator to the decision support system. Recommendations flow from the Decision support component to the Visualization and Display and back to the Simulator. Control commands from the user are received by the Visualizer and forwarded to the Simulator and the Decision support logic.

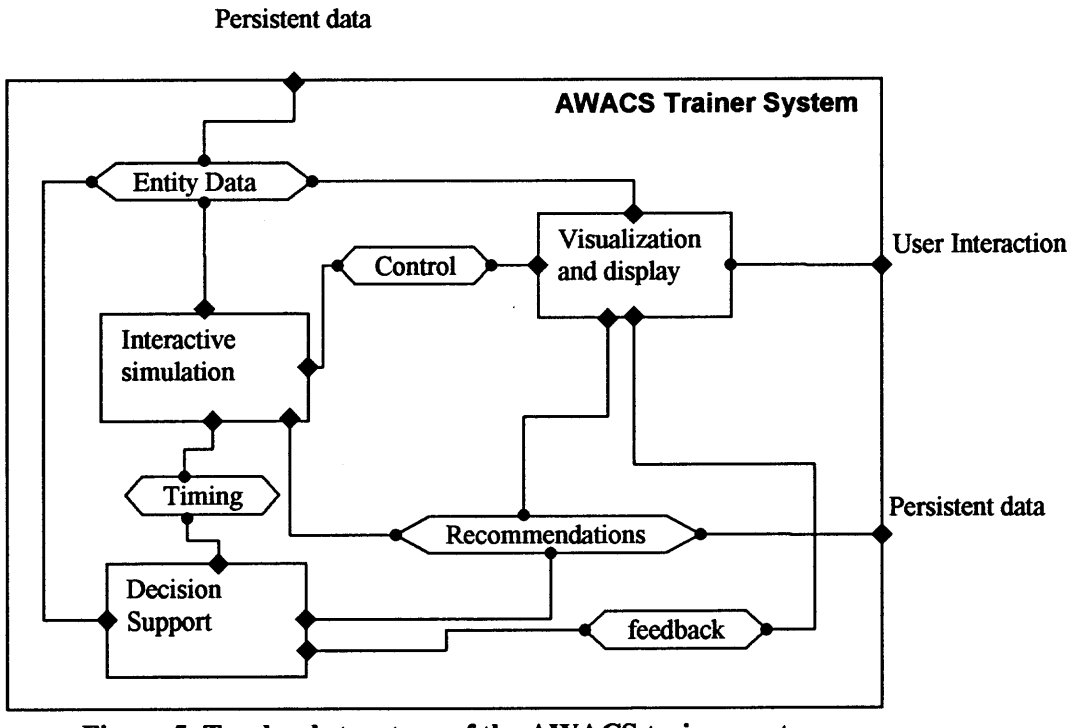

Figure 5. Top-level structure of the AWACS trainer system

Recommendations and entity data may be logged to persistent data modules, not shown as part of the system. The key of satisfying our objectives is the highly synchronized interaction among the three main modules. To ensure quality of service, special timing control must be provided, to ensure that decision support algorithms operate within the

timing constraints required by the simulator. The visualization component is not directly dependent on the timing control in order to increase the flexibility and scalability of the system, however the visualization processing must be competed within the time between Entity data updates, or the performance will suffer.

In the next diagram (Figure 6) we introduce the main conceptual components of the DSS subsystem. The Situational assessment component continuously monitors the location, speed and heading of friendly and enemy entities and maintains threat estimates. Based on capabilities, threat assessment, and available time, the Assignment generator performs resource allocation and produces assignments for Recommendations. Recommendations are usually of the form R_1->R_2, indicating the allocation of R_1 to R_2, either by a targeting, refueling, or any other order. Occasionally, more than two resources may be referred to in a recommendation, such as in a join order, where multiple resources form a group. The multi-objective evaluator performs heuristic optimizations to validate the allocation produced by the Assignment generator against a set of prioritized objectives. Feedback is provided to the Assignment generator, as the process may take multiple iterations. The final allocation and a set of Recommendations are passed to the WD recommendation distributor, which deconflicts and assigns individual recommendations to the command team members.

Both the Assignment generator and the Multi-objective evaluator have high complexity and may run in exponential time if special care is not taken. We introduce Timing

control, which ensures that the critical components of the DSS subsystem will execute within their timing constraints. This is discussed in more detail in Section 4.

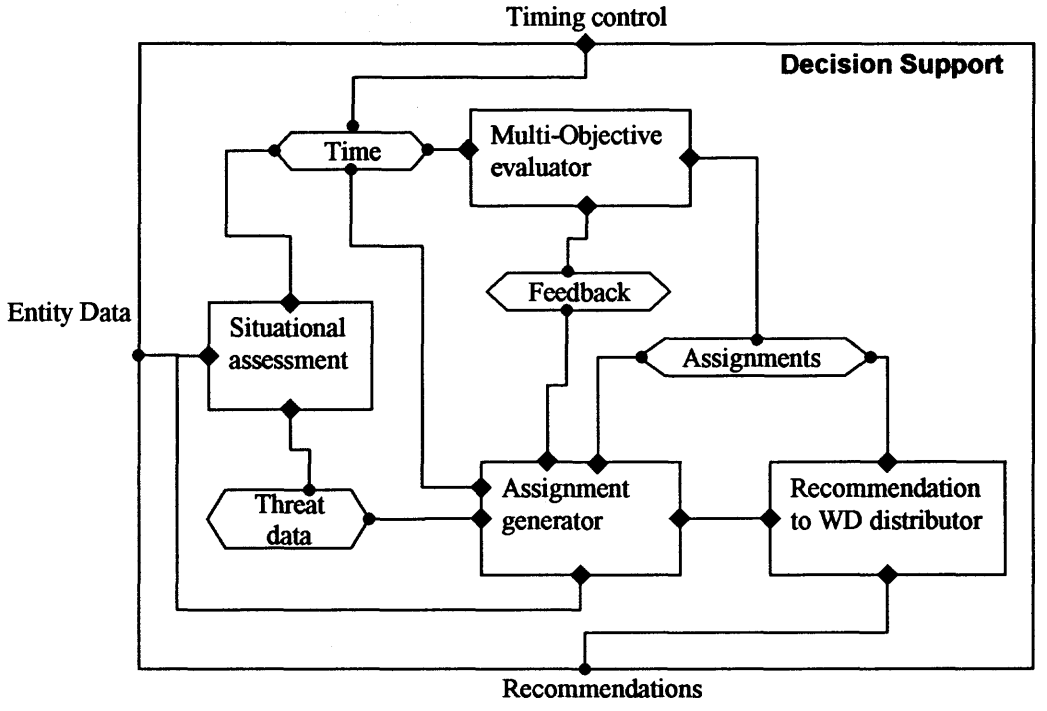

Figure 6. Decision support components and their interactions

3.2.3 System Layers and Components

Now we focus on the component packaging in the system. Since the backbone of the system consists of a distributed interactive simulator and decision support agents, an efficient distributed computing model is a key requirement. We reviewed three possible distribution models as outlined below.

Client/server model. This well-known paradigm is based on a centralized control by a "master" computing node – the server – and a number of dependent nodes – the clients.

In this model most of the computing takes place at the server node, while the user interaction is usually distributed among the clients. The clients and the server must be synchronized, though this is fairly easy, since all data and control takes place at the server. The model is fairly straight-forward to implement, though it does not scale well for large number of clients.

Peer-to-peer model. In this model all computing nodes have similar capabilities and each is executing part of the task. Essentially, each node provides equally well computing power, user interface, and communication capabilities. This model solves the "server bottleneck" problem in scalability with the client/server model. The challenge here is to keep a coherent, synchronized view of the problem space, distributed among the peers. This may require elaborate coordination schemes and large amounts of communication.

Three-tier model. The latest distributed computing model is roughly based on the client/server model, though it is enhanced by an intermediate level of services, connecting a remote data source to a remote client. This model is much more flexible than the traditional client/server, but still provides structured and protected handling of encapsulated data.

We selected an enhanced client/server model, which is, in fact, closer to a three-tier architecture, with the restriction that the data repository and the application layers reside on a single node. In Figure 7 we present the main components and their interactions. The Server and Client Layers exchange information via several common data formats, defined

in the Data Layer. This enables the development of multiple clients, participating in the same interactive simulation, but presenting and reacting to the data in different ways.

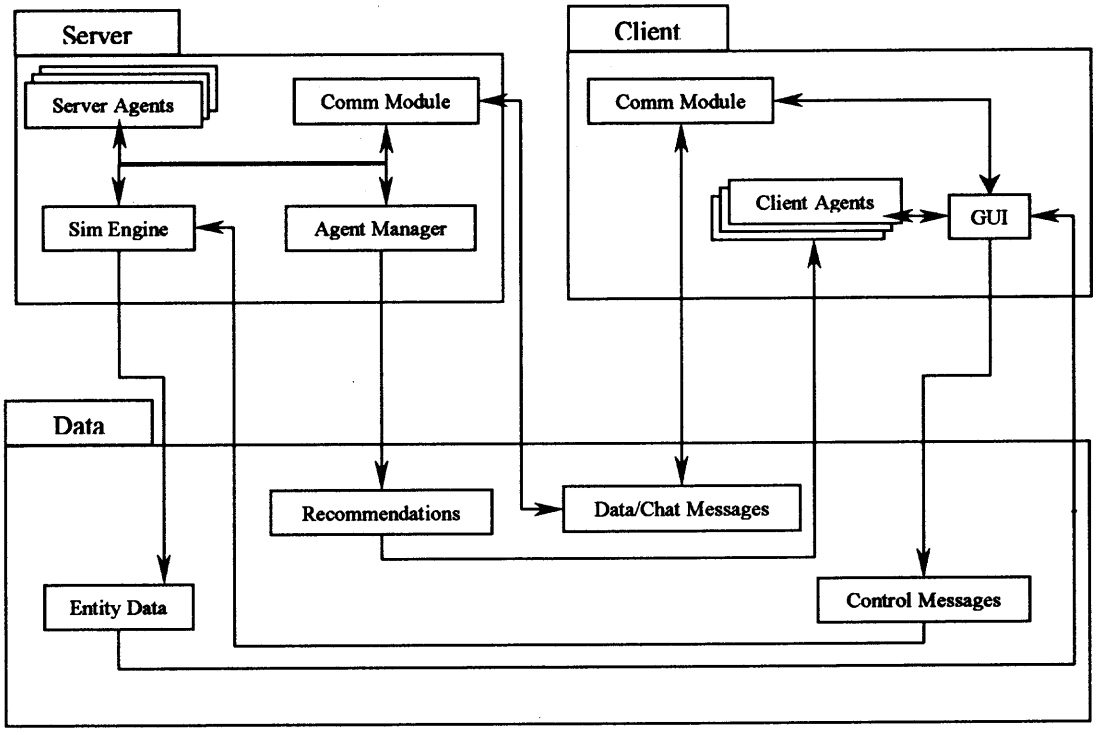

Figure 7. The layers and components of the system

The Server Layer consists of four major components: a Simulation Engine, an Agent Manager, an array of Server-side Agents (various types), and a Communications module. The Simulation Engine and the Agent manager are tightly coupled around common data structures and represent the core of the Server Layer. Various agents can be used in conjunction to the Agent Manager to perform autonomous functions, such as monitoring, evaluation and alternative planning. These agents should be re-configurable in real-time; one should also be able to construct new agents, conforming to the established interfaces to the Agent Manager. The individual agents may be instantiated per user or per

simulated resource, or they could be roaming, without attaching to a specific user or resource.

The Client Layer hosts all components responsible for interactions with the user and rendering of the tactical view. The GUI component (which internally consists of multiple sub-components) interacts with the user by displaying maps, icons, shapes, various choices, buttons, and selectors and by receiving input from the user in the form of various mouse-click combinations and keyboard activity. Since user interactions is encapsulated in this module, one can easily add alternative visualization and interaction models, such as VR displays, voice generation and voice recognition, data-glove input etc. The GUI receives its inputs from the Client-side Agents, the Client-side Communications Module, and from entity data, broadcast by the Server. The Communications module handles incoming and outgoing data and voice/chat messages. It may perform filtering on selected communication channels, as specified by the user. The Client-side Agents, similarly to their Server-side counterparts, perform autonomous functions, which are related to specific user. These may include performance monitoring and evaluation; rendering, interaction and history tracking of recommendations; user guidance and help, etc.

The Data Layer encapsulates the various entities used for communication between the Server and the Clients. All such structures, along with the methods of their transfer over an interconnection medium are defined here. The fact that the information will be communicated over a network implies that all data structures must be "serializable" or

that a one-to-one mapping exists between a data instance and a string of bytes, which encodes the data instance. Also, methods for mapping these serializable data structures to the ones used in the Simulation engine (not necessarily serializable) must be defined here. The set of data structures shown on Figure 7 includes Entity data, Recommendations data, Data/Chat Messages, and Control Messages. Entity data and are broadcast by the Simulation engine on every simulation cycle, in order to update all clients with the latest position, heading, status, etc. data of the simulated resources. Recommendations data is sent by the Agent Manager to specific clients, based on the role of the human logged in at that client. For example, recommendations sent to the SD station will be different than those sent to WD-1. Control messages usually flow from the clients to the server (unless a control message has to be forwarded from one client through the server to another client). Data/Chat messages are intended to connect two clients. Currently, these messages are routed by the server, though there is no reason for the clients to be able to communicate directly (given that they know each other's address).

We will now focus on the distribution of the system components to computing nodes. The diagram on Figure 8 shows the relations among implementation-level modules at the server side – on the left – and at the client side – on the right. The server and client are connected via Java's simplified version of an Object Request Broker (ORB) – the Remote Method Invocation (RMI) services. RMI provides a remote object naming and location service (the RMI registry). Once a remote object has been located through the RMI registry, its methods may be invoked, though special care should be taken to pass

only serializable parameters. Thus, the Data Layer from Figure 6 is utilized, even though it is not shown on Figure 8.

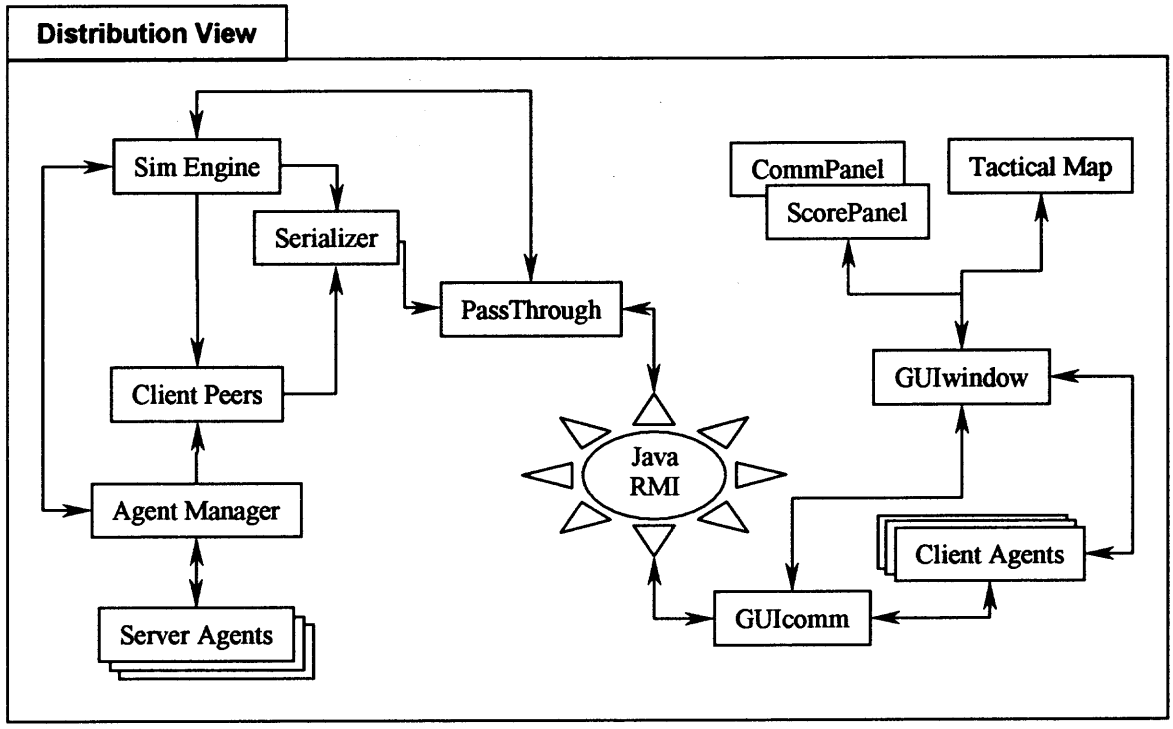

Figure 8. Distribution of components view of the system

All remote communications are controlled by two interfaces – the PassThrough on the server side and the GUIcomm on the client side. The PassThrough handles requests from the Simulation Server either directly or through the Serializer. The Serializer maps complex simulation entities and other data structures to their serializable counterparts. On the server side, each client is represented by a Client Peer, which is used to communicate with a specific client – either to update its entity representation or to send recommendations to the WD logged in at that client station.

On the client side, the GUIcomm handles all incoming (updates) and outgoing (control) requests for communication. The GUIcomm interfaces with the GUI Window and with the client-side agents. The GUI Window then passes all updates to its components – the Tactical Map and various information panels. Any user interactions are propagated back from the GUI Window elements to the GUIcomm. The client-side agents also can request communications to the server, in order to keep the agents at all sites synchronized.

Section 3.3 Management of Performance-aware Agents

3.3.1 Sharing Information

In a distributed interactive decision support system information management plays a key role to efficient and effective execution. Information about the simulated (or real) entities has to be up to date, coherent across all users, agents or humans, and easily accessible to everyone that needs it. On top of these functional requirements, there are usually demands for open and scalable design, both in terms of adding entities and adding new computing nodes, high performance and, often for reusing legacy data. Our design addresses the functional requirements and attempts a best effort heuristic optimization on the non-functional requirements. The rationale we used was based on our early design decision to differentiate between the Simulation engine and the Agent manager logic. Ultimately, the Agent manager should be usable with any (other) simulation engine, as long as the required information about entities and environment is made available.

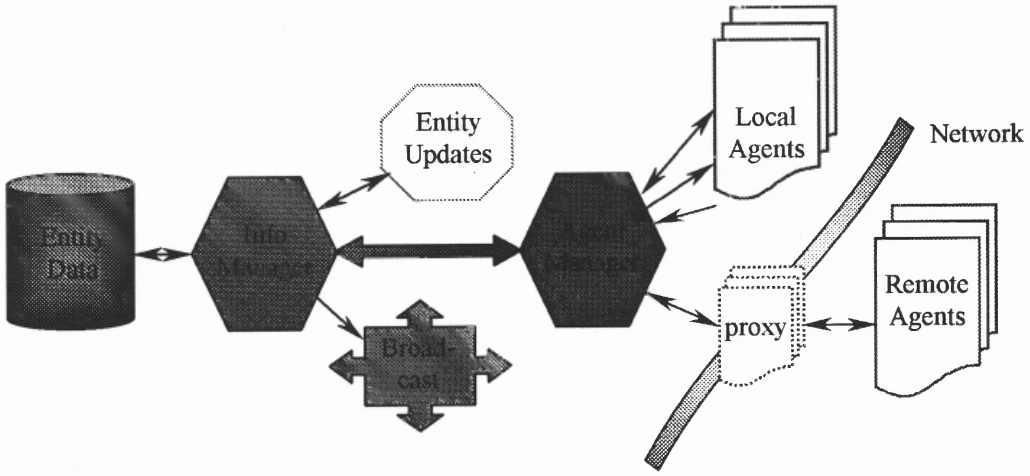

Figure 9. Data paths and information flows

Figure 9 outlines the flows of information in the system. Entity and environment properties are stored in a data repository (an object oriented database or a set of hash tables) on the server side. The repository is interfaced by Information manager, which provides consistency and security checks and subscriber/broadcast services. The Agent manager (which also resides on the server) taps directly to the Information manager, providing efficient and coherent view of the available data to the agents. The Agent manager exercises additional security checks, based on the types and permissions of the agents requesting information (e.g. friendly Monitoring agents are allowed only limited view of the hostile entity properties). Agents residing locally on the server are allowed to interact directly with the Agent manager – they can rear, write or modify entity data in shared data structures, provided they have the required security permissions. Remote agents need to exchange data with the Agent manager through proxies on the network. The proxy is responsible for packaging the data and handling the transmission detail, including evaluation of the network delays. The remote agents see similar data structures

to those presented to the local agents. If the data cannot be transmitted on time to the remote agents, the Agent manager is notified with a control message. The Agent manager then decides weather to initiate a re-transmit or to wait for more recent data, based on the type of data and the type of remote agents involved.

3.3.2 Synchronizing and Deconflicting Among Agents

When multiple agents are assigned to a single user (a WD or an SD), it is imperative that the advisories and recommendations generated are cross-referenced and deconflicted for the human to perceive a coherent picture. Such inter-agent coordination is performed by the Agent manager.

The Agent manager maintains a set of user profiles to match individual requirements and preferences with agent capabilities. When agents are assigned, enabled, or disabled by a user, the corresponding profile is updated. Thus, the Agent manager has current information on every user and is able to coordinate the efforts of all agents to satisfy each user's particular requirements. Matters could be more complex when the set of agents belonging to one user are physically distributed. Indeed, the user interacts mainly with the client/GUI layer of the AWACS tool, which hosts several agents on the client side. Thus, the Adviser and Evaluator agents on the client side have to keep up with the recommendations provided by the Monitor and Analysis agents on the server side. Such interactions are enabled via the agent proxies – the local representations of remote agents – and consequently via the centralized Agent manager. One such exchange occurs at the

point of recommendation acceptance. First the Analyzer or Monitor agents provide a set of recommendations for the WD they are assigned to. The recommendations are sent to the Adviser agent, who is responsible for interacting with the human WD. Suppose the WD decides to accept only one of the recommendations displayed for him by the Adviser. Within one simulation tick the adviser has to complete the interaction with the user (confirm the choice), and inform the Server side agents (Analyzer and Monitor) of the human choice. A synchronization between the Adviser and the Analyzer agents is needed, in order to prevent the analyzer from issuing the same recommendation on the next simulation tick.

The synchronization and deconfliction of recommendations from multiple agents to multiple users (both WDs and SDs) can be more complicated. Such recommendations may include resource handoffs – transfers of resources from one WD to another to balance the available resources. The Analysis agents, by design, first optimize locally the allocation of resources, belonging to the corresponding WD. After the local optimization cycle, if there are remaining unhandled threats, each Analysis agent optimizes the global allocation problem. At the end of this stage, the recommendations from all Analysis agents are cross-referenced by the Agent manager for inconsistencies or conflicting requirements, and non-conforming recommendations are dropped. At the next stage of the multiple handoff procedure, the recommendations to borrow or lend a resource have to be presented to the involved WDs and SDs. At this point, as many as six agents per handoff may be involved in a complex handshake and negotiation of the handoff.

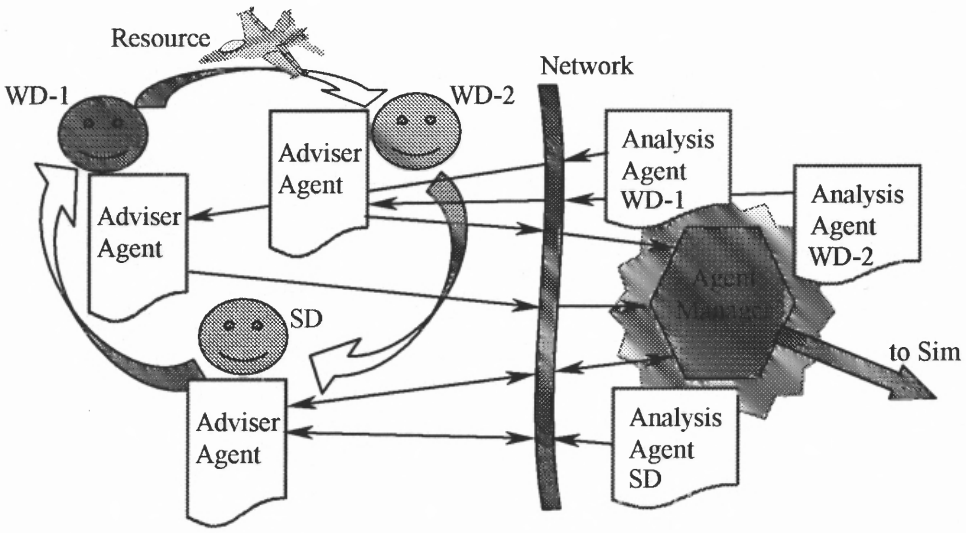

Figure 10. Inter-agent synchronization during a resource handoff process

Let us consider the handoff negotiation shown on Figure 10. We start with a handoff recommendation from the Analysis agent of WD-1. The recommendation is transmitted over the network to the WD-1's Adviser agent, where the human (presumably) accepts the recommendation. Next, the agent manager is notified that the handoff recommendation is accepted, which triggers a recommendation to WD-2 to accept the resource. The recommendation to WD-2 is transmitted over the network and the corresponding Adviser agent handles WD-2's response (acceptance or rejection of the resource). Simultaneously, the SD is advised that a resource handoff is initiated by WD-1, and similarly, SD's Adviser agent handles the approval or rejection from the SD. Finally, the Agent manager collects and processes the results from the three Adviser agents and if the handoff has been approved (according to the chain of command precedence of the SD over the WDs), the Agent manager informs the Simulation engine that the resource is now transferred to WD-2.

Section 3.4 Performance and Timing Constraints Analysis

There are two types of constraints in the AWACS environment. The hard real-time constraints are due to the fact that information on entities is updated periodically, as the Early warning radar sweeps the target area. These updates typically occur once every ten seconds (however the frequency should be re-configurable). Naturally, all processing involving the entity updates and visualization is periodic, with each period's computation bounded by the refresh time.

The second real-time constraint is dictated by the need of timely response to a changing tactical situation. Even though the processing associated with the decision support subsystem is not necessarily bounded by the entity refresh periods, it is quite reasonable for practical purposes to assume that any recommendations issued, based on "old" entity information will have little or negative value. In the AWACS context, given the speed of most entities, "old" means from a few seconds to a minute old information (or one to six refresh cycles at 10 seconds per cycle). Alternatively, it is conceivable that a incremental decision support is provided, where old information is retained and updated gradually as entity refreshes arrive. Incremental systems, however are less flexible and tend to omit threats associated with a newly detected enemy resources (such as a pop-up HTLAM). Thus we focus on a non-incremental design and we impose a hard deadline of N refresh cycles for the DSS subsystem, where N is a constant (between zero and six, usually two).

In Figure 10 we see a partial capture of the complex interactions between the Simulator, DSS, and the Visualization components of the AWACS system. The horizontal lines represent the refresh period boundaries of the simulator. At the end of each refresh period the new entity data is made available to the rest of the system (possibly distributed over a network). Such updates are synchronous and also serve as heartbeats of the system.

Depending on the timing constraints selected for the DSS subsystems, recommendations must be generated within a constant time N (N=2 refresh cycles in the Figure) after the initial refresh period. Thus the DSS process has a period, multiple of the simulation refresh period.

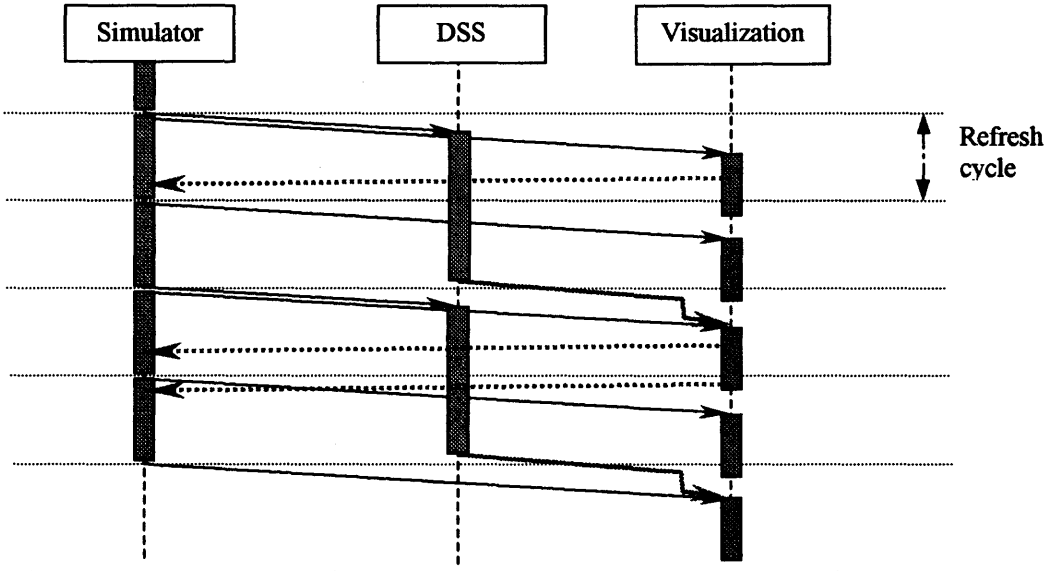

Figure 11. Timed interactions among the subsystems of the AWACS trainer

The Visualization subsystem, though lagging behind the simulator in absolute terms (due to propagation delays), has the same period of update, so the user perceives a consistent

picture of the universe. Recommendations are also displayed synchronously, in the beginning of every (N+1)st visualization cycle. Note the buffering of recommendations (the red arrow in the Figure) to accommodate the delay to the next visualization cycle – recommendations will not be shown in the middle of a visualization cycle, but only at a cycle boundary, in order to avoid confusion with previously issued recommendations.

The user interaction with the Visualization subsystem, however, is asynchronous. To support the interactive simulation, the user commands and directives must be accepted and processed asynchronously (the blue arrows on Figure 11). The challenge is to provide immediate feedback to user requests and still keep the timing requirements of all tasks. This is handled by spare processing cycles allocated to threads at the simulator node. Though the system will not guarantee to meet hard real-time constraints under user overload, it will monitor the excessive cycles needed to support user interaction and will self-adjust, to ensure graceful degradation.

CHAPTER 4

REVIEW OF RELATED WORK

Section 4.1 Agent Architectures

Distributed Object Approach. In the paradigm of object-oriented programming (OOP) languages, distributed computing is usually supported by language extensions for distributed objects. While standard OOP languages are quite powerful in defining control and data structures for single processors, most of them do not feature mechanisms for defining and implementing various object distribution models. Thus, with the development of networked and distributed computing, various extensions and models were developed. Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) is now considered a standard object distribution model. CORBA defines object location and method invocation services for remote (as well as local) objects, which supports transparent distributed interactions. Multiple implementations of CORBA and other Object Request Brokers (ORBs) exist, for a variety of languages and operating systems. CORBA's advantages of transparency and wide interoperability come at a high price of considerable overhead and relatively low (and unpredictable) performance, partially due to inefficient implementations and partially to CORBA's inherent architectural complexity.

Microsoft introduced its own approach to distributed object oriented computing – the Distributed Component Object Model (DCOM) – an extension of COM, which defines

services and interactions with remote COM objects. While DCOM addresses some of the performance issues, its availability for platforms other than Windows/PC is limited. The model is inherently dependent on Microsoft-specific features and is thus hard to analyze with external tools.

Sun Microsystems's Java was designed and built as an advanced network-friendly OOP language with built-in possibilities for extensions. Java's version of distributed object support, the Remote Method Invocation (RMI), borrows form the ORB model and from the older, well understood Remote Procedure Call (RPC) model. RMI provides remote object location services and a skeleton-stub implementation of RPC. It is more efficient, more flexible, but less programmer-friendly than CORBA or DCOM.

Another fairly recent approach, to distributed computing, based on *Mobile Agents*, has emerged from the distributed OOP models. Mobile Agents (a.k.a. Mobile Code, Mobile Objects) are autonomous or semi-autonomous software components, which can execute on one node, travel over a network, and continue execution on another computing node. Examples of fairly mature mobile agents systems include IBM's Aglets, Mitsubishi's Concordia, and ObjectSpace's Voyager. Most of the efforts in this field have been centered around Java or other similar network-friendly languages (e.g. TCL/TK). This approach benefits form high parallelism within a limited bandwidth network (moving small code to bulky data). The mobile agent approach enables easy construction of distributed network applications which require interactive, user-tailored computing. The disadvantages of mobile agents include little coordination support from the environment,

rigid dependency on the language and run-time support, little flexibility in the specification of agent movement.

Various groups have developed and applied more generic distributed computing methodologies, which are relatively independent from the underlying hardware architectures. These include the *Blackboard* approach, in which a common logical space is used to store and exchange data. The data items, usually referred to as *tuples*, provide communication and synchronization primitives. Prime examples of this approach are the LINDA environment (Gelernter93) and FliPSiDE (Schwartz95). Blackboard architectures provide a flexible framework for distributed computing by eliminating the tightly bound interaction links that other technologies require during interprocess communication. This is a double-edged sward though, because the framework does not provide the ability for explicit references to cooperating processes, and this could be practical in many cases.

Several research communities have modeled distributed computing by studying communication and coordination mechanisms among autonomous software entities, or Agents. Agent-based computing focuses on the interaction mechanisms among agents, which permit a rich set of coordinated activities. Effective models of interaction require the following basic capabilities: 1) a transport mechanism to convey messages in an asynchronous fashion, 2) an interaction protocol, defining the available types of communications and their semantics, 3) a content language providing the base for composition of requests and their interpretation, and 4) an agreed-upon set of shared vocabulary and meaning of concepts (often called on *ontology*). The most common

foundation technology used for such agent-based architectures is the Knowledge Query Manipulation Language (KQML) (Labrou and Finin, 1997). KQML specifies interaction protocols by defining symbolic *performatives* to represent information about the purpose of a communication. Since it uses a standardized representation of conversational interactions, KQML is limited by its reliance on a fixed set of atomic performatives. Arriving at just the right set of performatives in the ontology has been a major hurdle in this and other approaches.

Another approach to implementing the fundamental capabilities for Agent-based computing is structuring the agent's activities around the concepts of Belief, Desire, and Intention (BDI) (Rao and Georgeff, 1995). While BDI's emphasis on a higher level of abstraction has been important in giving direction to work on agent-based systems, its applicability may be limited by the structural requirements posed on individual agents. BDI makes stronger assumptions about the knowledge availability and processing within agents, which induces difficulties in operating with largely-legacy systems.

The Open Agent Architecture (OAA) ([MCM98]) also provides a framework for Agent-based computing. OAA focuses on a balanced set of objectives, including efficient interoperation, autonomy, coordination, flexibility and extensibility. The architecture incorporates a promising set of new technologies, though it is still under development. Practical applications of OAA will require improvement in scalability and robustness, as well as the construction of new development, testing and profiling tools.

We provide an overview of the related approaches to distributed computing in Table 1 below.

Model \ properties	Representative products	Technology	Description
Distributed Objects Approach (DOA)	CORBA (OMG), COM/DCOM (Microsoft), Inferno (Lucent) Java RMI (Sun), RPC (...)	Messages, RPC	Object request brokers (ORBs) dispatch messages to remote objects to their location on the network. The equivalent of RPC in the OO world. Powerful but inflexible and with high overhead.
Mobile Objects Approach (MOA)	Aglets (IBM), Concordia (Mitsubishi), Voyager (ObjectSpace), Plangent (Toshiba)	Messages, mobile code	In contrast to DOA, objects are free to roam from node to node. More flexible, but still high overhead. Imposes tough structural requirements on legacy code (serializable, mobile).
Blackboard Approach (BBA)	LINDA (Gelernter) FliPSiDE (Schwartz)	Shared memory model	Information is stored in "Tuple-space", accessed symmetrically from all computing nodes and tasks. The tuple space serves both as communication medium and as coordination mechanism.
Multi-Agent Approach (MAA)	Open Agent Architecture, InfoSleuth (MCC), Bee-gent(Toshiba), RETSINA (CMU)	Events, messages, mobile code	Inter-agent coordination is the key. Agents conform to an architecture, which defines the possible interactions among them. Events, messages, and triggers are often used to describe the interactions in an Agent Communication Language (ACL). Very flexible and scalable. Requires "agentifying" of legacy code.
Knowledge-Based Approach (KBA)	KQML/KIF, BDI, FIPA	Knowledge Bases, Inference engines	The Knowledge Query Modeling language defines "performative" phrases based on an application-specific "ontology". Agents communicate by means of requests and answers, formulated in KQML. The Belief-Desire-Intention model focuses on a higher level of abstraction. Agents coordinate based on their internal models of the world (belief) and formulate communications

Table 1. Approaches to distributed computing

Section 4.2 Resource Allocation, Real-time Constraints and Complexity

Our work is related to various efforts in resource allocation and other similar tools, in mobile code management, and in specialized compilation/interpretation. Some of this related work may address real-time or agent-based software, but likely most, at this stage, does not provide detailed application architectures. We do expect more of this work, by us and by others, to address decision support and other real-time applications that run on networked platforms (over the Internet or over a secure QoS-enabled network) in the near future.

In mobile code management, we credit Java's bytecodes [G95] with the first reported use of typestates [SY86], standardized scalars, and a trusted interpreter in a major commercial language. Not surprisingly, much of the current work in this area is therefore Java-related. NewMonics, Inc.'s ongoing work was the first to aim at a clean-room real-time Java implementation (with some language additions), featuring predictable garbage collection. Many established vendors, including Hewlett-Packard, Sun Microsystems and others, now claim and offer products with the same capability. An extension of Java called Sumatra [ARS97] features a distributed monitor that helps Sumatra programs adjust to resource availability. Support for Java agents is often provided, through research efforts (e.g., Jada [CR97]) and to some extent, commercially (e.g., ObjectSpace's 1997-announced Voyageur). These efforts do not appear so far to consider tradeoffs among non-functional objectives.

Omniware [ALLW96] combines the use of software fault isolation and careful high level language to detailed-level (including RISC instructions, registers) intermediate representation translation to add reasonable safety while not loosing too much efficiency in mobile code. VCODE [E96] also uses a detailed (i.e. low-level) even if machine-independent interface (of an idealized load-store RISC machine), and very efficiently generates dynamic code. Clarity Mcode [LDG95] is a retargetable intermediate representation for compilation on both Sun and non-Sun platforms of a simple C++ dialect (Clarity C++) developed by Sun. Auslander et alii [Aetalii96] applies carefully incorporates (through templates, setups and specializations, and other methods) local optimization into dynamic code generation. These efforts appear to rely on ad hoc allocation and management.

Software architecture approaches to tools, such as the DARPA-funded Honeywell HTC work [BV95], aim to support formal model co-generation and do some behavior prediction. However, the work does not appear to support multi-objective or mobile code systems. Two related DARPA projects at Honeywell Space & Missile Systems (with HTC cooperation) do aim to eventually produce a tool for parallelization and possibly allocation in HPC systems. However, neither mobile code nor multiple objectives have been considered. Another HTC tool [B93] does perform ad hoc allocation while instrumenting low-level computational requests. Among commercial tools, Telelogic's SDT and NuThena's Foresight do perform allocation-like design, but no allocation for HPC or any parallel or distributed platforms per se. To the best of our knowledge, there are no allocation tools for mobile code components (even *conventional* program

development tools for Java and JavaScript are in the emerging stages as of this writing). Some interesting research on new design paradigms for relocatable and mobile code (e.g., [BC95, BGP97, CPV97, GV97]) are emerging as well, though considerably more work will need to be done.

In addition to the work in tools and mobile code management, there are notable efforts in related research areas, including security for agents [FGS96, G96], mobile object systems and their support [D97, F97], partial information query in databases [BDW92, GJ95, SMF92], incremental mechanisms [H92, P95], static analysis and transformations [MR90, SGL95], programming environments and attribute grammars [R88], graph algorithms, especially transitive closure and topological sorts [P95, Y93], view materialization for databases [BCL89], incremental and automatic program derivation [LT95, JGS93], finite differencing [PK82], and incremental languages [YS91]. A number of recent efforts are also noteworthy, trying to apply well-developed theories of program correctness to limited tradeoff considerations. Among these Necula [N97] attaches proofs or proof obligations to libraries or foreign code; verifying the proof then adds to the trustworthiness of the code. Plezbert and Cytron [PC97] analyze when or whether to compile ("just-in-time" or "better-late-than-never") code elements. Hogstedt, Carter and Ferrante [HCF97] improve predictably the parallelization of tight nested loops, which is relevant to analysis of real-time programs.

While these and many other advances are very commendable, they do not sufficiently address performance tradeoffs among mobile code objectives (e.g. compilation vs.

interpretation vs. safety). Moreover, many cited systems have a strongly "pre-wired" notion of how to optimize and do not allow for flexibility nor dynamic decision-making (and unmaking). Finally, many of these systems are ad hoc (e.g., VCODE is efficient but manual and thus, in a sense, ad hoc). Our approach, through an integrated tool family, automation, and an orthogonal treatment of conflicting objectives and mobile code management, should thus be of interest and contribution the existing state of the art.

CHAPTER 5

EXPERIMENTAL PROTOTYPE

Section 5.1 An Agent-based Decision Support System

The prototyped test-bed environment, under development at 21st Century Systems, Inc. provides the basic simulation functionality for decision support system development. The prototype originated as an AWACS commander (Weapons Director) team support system and evolved as Joint Forces command and control distributed simulation and decision support environment. The simulator features an array of airborne, ground- and sea-based platforms, weapons and systems. Realistic command structures, chain of command, rules of engagement, failure modeling and detailed physical models bring the test-bed fairly close to a state-of-the-art high fidelity simulator, at only a fractions of both the cost and the required computing power. Developed entirely in Java, the platform provides the advantage of open-ended Object-Oriented design, relatively low system requirements, high portability, and easily networked modules. The test-bed hosts an advanced multi-agent decision support system. A variety of agents monitor and evaluate the current tactical situation and, based on heuristic driven resource allocation, provide real-time recommendations to human operators and commanders.

The human user is assisted by interactive software agents, which perform evaluations and make intelligent suggestions, at the problem domain layer (e.g., which class threats to engage first) as well as at lower layers (such as which allocation heuristic to pick, what

type of precision to sacrifice trying to integrate an objective function, and so forth). A slate of simulation, resource allocation, and other auxiliary tools support the environment's function. Libraries of algorithms and heuristics are used throughout the agent-based decision support environment.

The users are immersed in an uncluttered, interactive, responsive environment. Each user has access to a graphical or, ultimately, a visual, even virtual world. He or she can monitor individual resource's status and performance, or the entire team's (or any subset thereof) health and accomplishments as required. Communications among command team members locally or at different locations can take place explicitly or implicitly in this environment. Cooperation, coordination, collaboration, and inter-personal communication are supported in multiple modes. In this virtual environment, users may navigate (explicitly at the problem domain layer, and implicitly, in the implementation layers) through libraries of existing software agent specifications relevant to the problem domain, providing agent behavior descriptions, objectives, and constraints. The user is also able to create such agent specifications anew. By selecting different agent behaviors (e.g., choosing a heuristic target a high priority threat), the user engages the utility, cost, and objective functions engine to scale, fuse, convert and integrate individual cost functions into an objective and/or utility function. The user also engages the overall resource allocation engine to drive the allocation process, interactively, automatically, or in mixed mode, in accordance with the agent descriptions and the objective function(s), and subject to the constraints. The target resources are selected, on the basis of (partial) query criteria, user hints and automatic search and analysis. The user receives feedback,

projecting possible outcomes, success rate, and the extent of fit with the desired mission objectives, through plotted analysis or (real-time, animated) simulation. Colors, shapes, icons (and eventually sounds, video and live sensor feeds) are used to help the user appreciate the updated allocation and its properties.

Section 5.2 Test-bed Design and Implementation

A prototype of the decision support test bed has been designed and implemented. This software system integrates an extensive combat simulation with complex resource evaluation and allocation algorithms to model the dynamics of a modern theater of warfare. The combat simulation tool, paired with sophisticated and realistic software agent architecture, evaluates and visualizes the effects of decisions locally and to the global combat mission. An array of intelligent agents collects and filters information, evaluates the changing tactical situation, assesses available and committed resources and based on allocations and outcome predictions provides advice to the human commanders. In addition to its primary use as a decision support tool, the system can be used in both training and research as well as (eventually) for combat capability enhancement. Simulation, resource allocation heuristics and analysis, combined with intelligent agent technology are the key contributions of this prototype. We base the design and implementation of the simulation tool on our experience with other decision support and resource allocation software tools for large-scale system synthesis and command and control problems.

5.2.1 Environment and Communications Model

In a realistic combat mission, multiple forces combine their efforts for a successful outcome with minimal losses. Such complex missions must synchronize the activity of a large number of teams, possibly including Navy, Air Force, Marines, and Army forces. In our simulation tool, we currently consider Navy and Air Force combined missions, while we are extending the environment to be able to include all forces. We model and simulate a wide array of friendly combat units and groups and their interactions, as well as a large set of enemy units and groups. Among the entities we model there are Navy carriers, cruisers, submarines, carrier-borne aircraft, Airforce fighters, bombers, tankers, jammers, AWACS etc. Team participants communicate with each other and with their commanders to coordinate and implement the overall mission. Combat units may communicate directly or indirectly. Two units can communicate directly if they are siblings or superior-subordinate in the chain of command. Indirect communication may take place either via an intermediate unit (a commander) or through the environment (via signals). The communication links in our model abstract a wide range of possible communication activities.

Direct communication occurs between a superior and subordinate or between peers in the chain of command when information needs to be exchanged between the two (or more) units. Possible information includes orders to subordinates, confirmations and status reports to superiors, reports and requests to peers, etc. Communications take place over a communication channel or link. The link specifies a set of properties for the possible communications between the two endpoints. We consider the following properties:

Type:

(a) permanently open channel;

(b) sporadic;

(c) periodic;

Medium:

(a) radio - FM;

(b) radio - AM;

(c) light signals;

(d) internal digital bus;

(e) other;

Path:

(a) direct point to point;

(b) indirect, via intermediaries;

Attributes:

(a) range;

(b) bandwidth;

(c) uni- or bidirectional;

(d) security/encryption;

(e) fault probability;

Operator (or pilot) dependent:

(a) switched frequencies;

(b) busy/unable to communicate.

When a communication link is established between two entities, the link properties are determined completely according to the current tactical and physical situation (resources available, weather conditions, enemy presence, etc.). Some link properties may change dynamically in the course of the mission, such as range (dependent on the environment), fault probability (dependent on various fault-inducing factors), etc. Multiple communication links may exist between two entities in order to support fault tolerance or for increased functionality. Most communication links are typically bidirectional, where orders are passed form a commander to a subordinate while information and reports are communicated from the subordinate to the commander. The nature of some types of communication, e.g., via light signals, however, restricts the link functionality to a unidirectional channel.

Indirect communication can occur in two distinct ways: communication through intermediate entities, and communication via the environment. We are mostly interested in the first type, since it is essential for modeling and implementation of command and control features as well as for realistic battlefield simulation; we have a limited model for the second type. Communication through intermediaries is especially important for hierarchical chain of command implementation. Usually, a commander communicates an order to a subordinate unit commander and not to all unit members. Therefore, for successful delivery of the order, a communication path must be established from the order origin to every possible destination. If such path is not available for some recipients, the order delivery cannot succeed and its successful completion will be nearly

impossible. Communication path unavailability will result in communication fault. For example, if the commander of a fighter group is taken out, this will interrupt the established communication protocol with the remaining fighters in the group. To correct this, the second in command is assumes the group leader position. In this case, all communications to the group will be now routed through the new group leader.

The second type of indirect communication, via the environment, is encountered less often, typically in situations where the sender entity may or may not be aware of the possible communication. An example of indirect communication of this type is when a submarine leaves a sonic signature in its environment and another sonar-equipped craft is able to read that signature. Similarly, radar and infrared radiation could be considered indirect communication. Our model does not include full representation of such phenomena, however, we do provide basic features, which could be extended to support it.

5.2.2 Simulation Model and Simulated Entities

Our simulation engine is based on a rigorous mathematical model of the combat environment. A multitude of simulated entities along with their properties and interactions are described by equations and functions, which are dynamically evaluated during the simulation. In addition to the physical model of combat entities, we have developed a doctrine model representing chain of command and rules of engagement. A non-trivial resource allocation algorithm is applied in simulating and evaluating combat

actions of both hostile and friendly forces. A realistic fault model is used to simulate and evaluate the effects of sporadic and persistent failures to the general combat efficiency. We base our mission outcome evaluation and agent assistant tools on a fairly sophisticated model of the simulated battlefield. Nearly all aspects of realistic three-dimensional navigation geometry, including geometry of turning, climbing, intercept and pursuit are modeled by mathematical equations. A wide array of warships and aircraft, with their specifications and various armament, both US and foreign, including carriers, cruisers, submarines, fighters, bombers, tankers, jammers, AWACS, etc., are available for use in the simulation.

The simulated combat units are modeled as classes and objects forming an Object Oriented hierarchy. A class of moving resources parents a sub-tree of classes modeling air, ground, water, and amphibious vehicles. Further specialization includes different types of vehicles, such as fighters, command vehicles, carriers, etc. The leaf nodes in this hierarchy represent specific vehicles, e.g., an F15 fighter aircraft, an aircraft carrier, or a nuclear submarine. Each entity within the hierarchy has specific models of motion geometry, pursuit or intercept capabilities, weapon arsenal and carrying capabilities. The instance of such a leaf class is an object participating in the simulation, reacting to orders and interacting with other objects. For each entity, we maintain a dynamic representation of its perceived value, either as a friendly resource or as an enemy posing an immediate or future threat.

The simulation tool incorporates an extensive command hierarchy to represent the relations among senior commanders, unit commanders, and individual combatants. Software agents may be placed in positions of commanders and combatants eliminating the need for human operators at every position. Orders can be given either by human commanders, or by agents, via special communication channels, connecting the entities in the chain of command. We rely heavily on software agents to construct computer-generated forces (CGF) in our simulation environment. CGF are commonly used in training exercises to provide enemy and friendly resources for any particular mission. The agents controlling our CGF are capable of communicating with human or agent superiors and subordinates, receiving and carrying out orders and interacting with other entities in the simulation.

Our chain of command model includes human commanders/subordinates as well as software agents. The allegiance of units and their assignment to commanders is determined during startup in by the contents of a standard configuration file. This information is dynamic and may change in the course of the simulation, for example if an Airforce fighter is assigned to land on an aircraft carrier. Orders from commanders to subordinates and reports form subordinates in the simulation are conducted on communication links – special objects, which facilitate the transfer of information between simulation entities. Communication links are dynamic, as their properties may change or the link may fail altogether. Such failures can be introduced according to our failure model and are monitored, analyzed and displayed continuously.

5.2.3 Command and Control Model

The problem of command and control simulation is essentially one of resource allocation. Each commander is assigned a set of resources, such as battleships, aircraft, motorized units etc. and a current set of objectives or missions. The goal is to complete the missions or as many of them as possible, according to predetermined strategic priorities utilizing most efficiently and fewest resources as possible. Additionally to this local (per commander) optimization problem, there is a global resource-allocation problem existing among all commanders. Common practices such as transferring resources from one commander to another make the problem extremely tough – practically unsolvable in reasonable amounts of time. We approach this problem with a set of resource allocation heuristics, which drive the logic behind the Commander and Advise agents.

In our prototype we rely on few assumptions, which may be relaxed in the future. For example, a fighter group will always take any targets of opportunity. This might be tactically incorrect in some situations, however, it greatly simplifies the reasoning about possible outcomes for the Adviser Agent recommendations.

5.2.4 Fault Model

We consider faults on individual communication paths, faulty behavior of entities and regional faults that can affect a whole contiguous segment containing all entities and communication paths contained therein. For each case, we consider both transient and permanent faults. A permanent fault has a random time between two occurrences with a

frequency distribution and a mean at the user's choice. For the transient faults, in addition to time between two successive occurrences, we also need a random duration while the transient is active. Once again, the user can choose the distribution and related parameters. We have an initial model of related faults. Moreover, in our model a cascade of faults can be traced to its origin. Human-related faults can affect the system in two ways. Firstly, faults in entities and/or the communication links can cause faulty behavior. Various reliability and failure models can be used for this purpose. Secondly, operator/pilot errors may adversely affect the mission outcome. These can be accounted for in the same way as above.

We insert faults at active object entities and into communication links. There are two principally different ways to carry out simulation with faults:

1. The aspect of time between faults is not included in the simulation system. Instead we inject specific fault(s) and study the response of the system to these injected faults. This is akin the various schemes of fault/error injections used in fault-tolerant computing literature in order to study fault coverage and other fault handling aspect of system behavior and hence also to debug algorithms of fault-tolerance employed.

Or

2. We do include the aspect of time between fault occurrences in the simulation model so that reliability/availability/safety measures of the system can be estimated from simulation results. The price to be paid for this added

advantage is the slow down of the simulation runs and the number of simulation runs that will be necessary since rare events will now have to be accommodated.

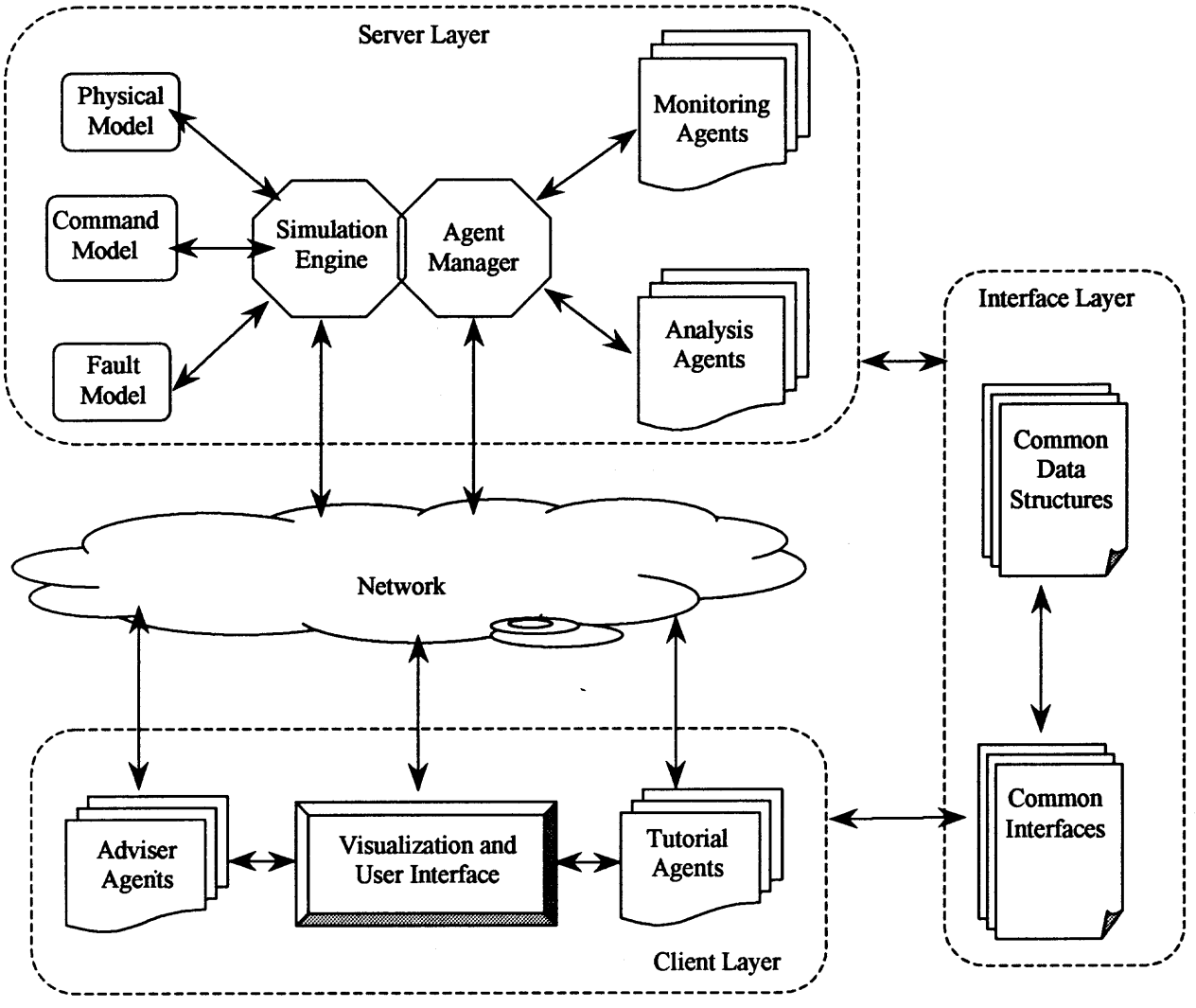

Figure 12. Distributed architecture overview

We have currently implemented the first option. Faults are injected either manually or automatically according to a fault distribution relationships. The effects of these fault insertions are later observed and studied over the course of the simulation.

5.2.5 Architecture of the Prototype

Figure 12 illustrates the distributed software architecture of our simulation and decision support environment. The Simulation Engine provides the base functionality of the test bed. The simulator consults the following models:

- Physical model, describing the behavior of all physical entities, such as ships, airplanes, submarines, ground vehicles, etc.;
- Command model, providing chain of command and rules of engagement notions;
- Fault model, providing the strategy for generation and evaluation of faults.

These tree models capture problem domain knowledge in the form of tables, algorithms and functions, facilitating accurate and realistic simulation. For example, suppose a Navy F14 jet is returning to its carrier. The physical model will be used to calculate the approach path, speed and angles. The command model will determine if the fighter belongs to the carrier, if it has the proper orders to land, and how will this affect its fighter group. The fault model will evaluate any damage and malfunction to the plane and its effects on the landing procedure.

An array of Agents communicates to the Simulation Engine through an Agent Manager. They monitor the strategic situation and provide the logic for automated enemy and friendly forces. These agents may substitute human commanders and vehicle operators

(pilots) and act on their behalf, issuing and executing orders. They also interact with existing human commanders by means of standard communication protocols and channels. The Simulation Engine, the Physical, Command and Fault models, the Agent Manager and the Analysis, Monitoring and Navigation Agents form the *server layer* of the simulator. The server layer entities communicate with the Visualization and User Interface module and with the Adviser Agents via a network connection. The Visualization and User Interface modules maintain coherent views of the simulated environment among all participants. They display the entities visible at a particular station for a specific human participant and interact by accepting orders from human commanders.

The Adviser and Evaluator Agents provide feedback and evaluation of the human commanders' actions. They interact with the server layer Agents over the network to maintain current their tactical position representation. The Adviser Agents evaluate alternatives to the human orders, which may achieve the same goal with fewer resources or in less time. They also consider the global allocation of resources and suggest transfer of resources from other commanders if needed. The Visualization and Interface module, along with the Adviser and Evaluator Agents form the *client layer* of our tool. The Visualization module is implemented in the client in order to reduce the load on the server. Each client is responsible for the graphical rendering and display of the user interface. This architecture also allows independent interfaces to different user, as well as taking advantage of heterogeneous computing environments.

5.2.6 Implementation Language

We have selected the Java language for our implementation, in order to support distributed heterogeneous computing platforms, at competitive cost, as well as to maintain an open architecture, which allows us to interface with a variety of external tools. While Java is a relatively young programming language, it has quickly gained a prominent place among other development platforms because of its network-friendly design and its syntactical resemblance to the C (and C++) language. With the emergence of Java just-in-time (JIT) compilers and optimizing full (binary) compilers, the improved performance combined with platform independence and ease of use (and reuse) of Java code will position this platform as a leader in the research and development of command and control decision support systems. Given our selection of the Java language, we chose to implement our distributed computing mechanisms through Java's Remote Method Interface (RMI). Even though RMI, as part of Java, will still evolve, it features a relatively simple implementation of a well-understood Remote Procedure Calling mechanism. We have successfully enhanced the test-bed from its original single-user implementation to a distributed, multi-user architecture, as discussed previously. We are now looking into possible interfaces with other tools, simulators and models, such as ModSAF, MathLab and others.

5.2.7 Visualization

Key aspects in the simulation interface and visualization are information management and information presentation. A hierarchy of increasingly more detailed information must

be utilized to manage the complexity and balance the volume and level of information. Presentation strategies show resources, event/fault history, and trends and predictions. Types of views include graph-oriented diagrams, matrix charts, timelines, and statistical plots. Performance management, visualization, and analysis strategies being employed in other domains (e.g., enterprise networks) are incorporating features similar to those in our battlefield simulation tool, such as change-point detection and performance agents (refer to appendix). Agents interacting with the simulation engine may assist the user and engine in managing the information. Adviser Agents may provide feedback to guide the engine or user regarding display options. Performance Agents may invoke special functions for reporting, data integration, database access, and so on.

The simulation GUI is the user's view of the simulation environment, providing both presentation and control functions. A top-level display identifies the status of the simulation engine and lists an array of options. This display may be customized to a particular user or user-group, so that only those options relevant to the user(s) are displayed. One group of users ("trainers") is able to parameterize the physical, command, and fault models in order to represent specific battlefield simulation conditions. Another option, available to all users, is to open interfaces to agents (e.g., Analysis and Adviser) to configure, query, or control them. The user may also enable visualization of the simulation, selecting from several visual displays of the simulation and of the current fault model parameters.

A typical high-level view is an animated system diagram, updated over time. Our System Diagram depicts simulation entities (nodes) interconnected by communication links (edges), drawn in the format of a graph. The graph is dynamic as entities and links enter/exit the battlefield. It is hierarchical and coded. Hierarchy is essential to managing the potential complexity of a large-scale environment having many resources. A node/edge may represent a collection of resources, which may be viewed individually by zooming in. The elements of the graph are color- and symbol-coded (using standard military symbology) to represent the state of the resource, such as: operating status, number of faults, type of fault, failure-related metric value (e.g., reliability, availability, etc.), performance metric value (e.g., bandwidth), failure-prediction indicator, etc. Elements of the graph may be hyper-linked to more detailed information. The Uniform Resource Visualization (URV) approach is directly applicable to the system diagram visual display.

Failure analysis and visualization focus on resource displays, metric and fault timelines, and statistical analysis and displays. Each display provides access to complementary views (having related information) and/or low-level views (having detailed information). A Failure Matrix display labels rows and columns with simulation entities, each label associated with its own cell. In addition, the cells of the matrix represent communication links between entities (in the corresponding row and column). The display is animated over time by color-coding the cells to represent a value of interest, such as: number of faults, type of fault, failure-related metric, performance metric, failure-prediction indicator, etc. The patterns of color-coded cells relate to failure patterns. Fault-related

and performance Metric Timelines coupled with change-point detection provide insights into periods of faulty operation and forecast potential problems. An Event Timeline that marks the occurrences of faults gives specific details over time. This timeline may associate fault markers with a resource URV. A statistical Scatterplot of event or resource attributes facilitates cluster analysis and highlights anomalous behavior.

Figure 13. Distributed scenario with Navy and Airforce DCA missions. The enemy attacks with Fighters, Bombers and Cruise Missiles.

The crew of an AWACS aircraft, and in particular the teams of Weapons Directors, perform the critical task of coordinating and allocating combat resources to pending missions. Due to the high criticality, the stringent time constraints and the inherent complexity of optimal allocation, the Weapons Directors teams are faced with an

extremely difficult task. The situation is only worsened by potential failures in equipment, communications or human fatigue. The quality of decisions, which is critical for successful mission completion, may be undermined by a number of objective and subjective factors. For instance, when a pilot belonging to one Weapons Directory is transferred to another, the WD who requests the transfer must get an approval from the Senior Director (SD), but not by the pilot. This situation creates tension since both the pilot and the SD are officers, but the WD is usually a NCO. The credibility of the request may be questioned, undermining the integrity of the team and reduces its efficiency.

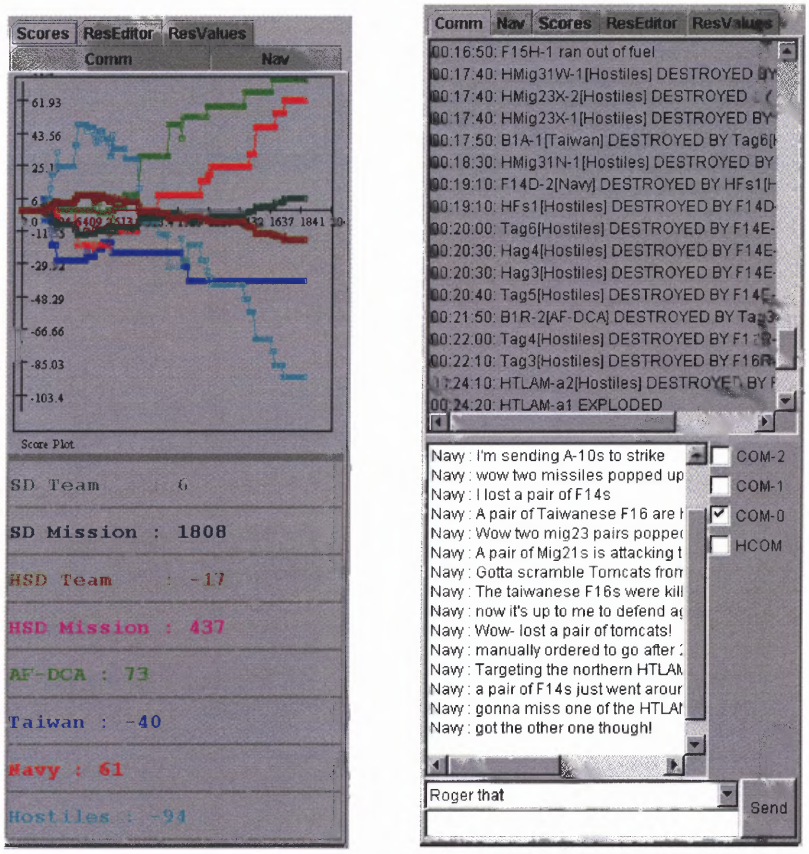

Figure 14. Performance scores plots and communications panel

The combat command and control simulation environment and decision support tools facilitate distributed training and improves the quality of mission-critical decisions made

by the team of WDs through analysis and isolation of possible failures and recommendation of viable alternatives. To illustrate the operation of the suite, we present an imaginary joint defense scenario involving this island of Taiwan off the Chinese shore (see Figure 13). A congregation of friendly forces, represented in blue, is stationed on and around Taiwan (Airforce bases) and off the shore (Navy carrier group – the USS Constellation). Our forces are threatened by multiple waves of Chinese fighters, bombers and cruise missiles, aiming to damage and destroy our critical resources (bases, carrier, AWACS, Hawkeye, Tankers, etc.). Figure 13 is a capture of the training mission in the initial stage of the scenario, when the friendly forces fight for air superiority, while defending their assets against high-threat attacks. We see the current position of the forces and several targeting assignments (indicated by blue and red lines). This scenario is designed for four commanders – two friendly defensive counter-air (AF-DCA and Taiwan) Weapon Directors, one friendly Navy Strike force, and one hostile Weapons Directors (controlling all enemy assets). The enemy attacks our bases and high value assets (HVA) with fighters and bombers. The goal of the exercise is to neutralize the attack while preserving as many friendly assets and destroying as many enemy assets as possible. A defense priority is the identification and neutralization of high threat hostile weapons, such as the two cruise missiles, HTLAM-1 (South-West corner) and HTLAM-2 (North-East corner).

Figure 13 shows the entire simulation window with its visual controls and interactive areas. The top of the window hosts the simulation control buttons, commander log-in and assistant agent buttons. Load and performance measures can be selected for display via

the drop-down list on the top right. Each director's performance is evaluated with score representing the quotient of achieved victories to the number of resources lost. The team performance is evaluated by two separate measures – a combined individual score (team score) and a multi-faceted integrated mission score. The main part of the simulation window is occupied by the interactive battlefield map (center-left). To the right of the map the user can alternatively display score graphs, current events, navigation tools (selected on Figure 13), failure management schematic, or failure distribution graphs. In Figure 14 the score graph (left) and current events panel (right) are displayed. Multiple tools panels are located on the left side of the map. These include selection and status verifications tool, zoom in/out, display properties selector, resource editor and others.

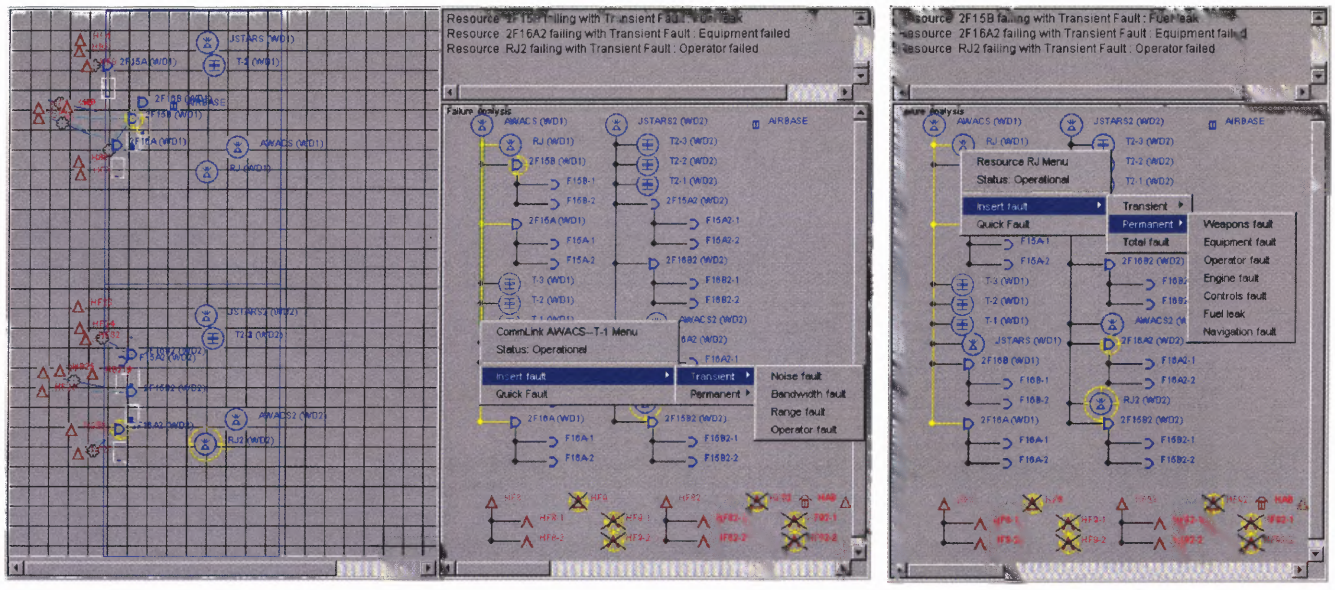

Figure 15. Battlefield map and failure management display. Multiple fault insertion pop-up menus enable user control over faults.

Failure management is facilitated by an additional interactive display, which can be shown or hidden, depending on the user level of expertise (e.g. it could be made available only to the leader of the exercise and not to all trainees). The display shows a schematic

of the resources in a hierarchical order, the communication links between them and failure status of both resources and links. All resources and communication links can be probed for the presence of faults, both manually, by human users, and automatically by agents. Failures can be inserted automatically, according to pre-defined rates and distributions, or manually, by a human director. When the failure analysis is enabled (Figure 15), the simulation window displays the failure management panel on the right. Failing resources are displayed with yellow circles surrounding their icons (see RJ2, 2F15B, 2F16A), while communication link failures are similarly visualized by yellow-colored lines. The pop-up menu, shown in Figure 15, for the link between the AWACS and tanker T-1, enables the user to insert transient or permanent faults at a specific link or resource. The failure management tool provides automated insertion of faults to all simulated resources or to a pre-defined subset. The automated fault insertion will conform to a specified distribution pattern. The fault insertion rate and pattern can be visualized by various plots. Figure 16 shows a plot of the number of transient faults over time.

Based on tactical situation assessment and on current fault status, the assistant agents provide recommendations and alternatives for team of Weapon Directors. The recommendations, shown on Figure 17, may be accepted by a WD, in which case an order is issued, or may be replaced by a manually entered order (through the VUI). Each recommendation is backed up by bias-free agent-generated rationale, substantiating the validity of the actions recommended for the current situation.

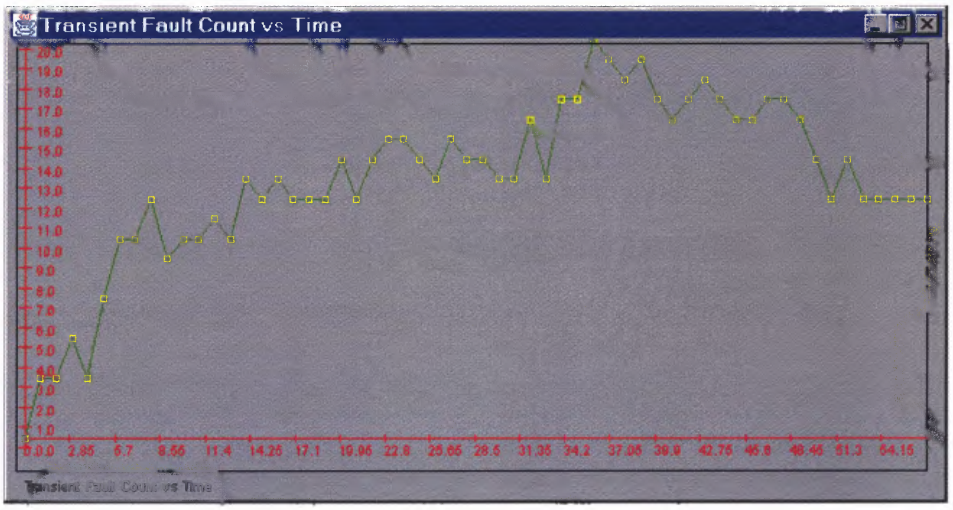

Figure 16. A plot of the number of transient faults over time (in seconds) The number of faults fluctuates around a pre-set value, reaching a peak at the heaviest enemy attack.

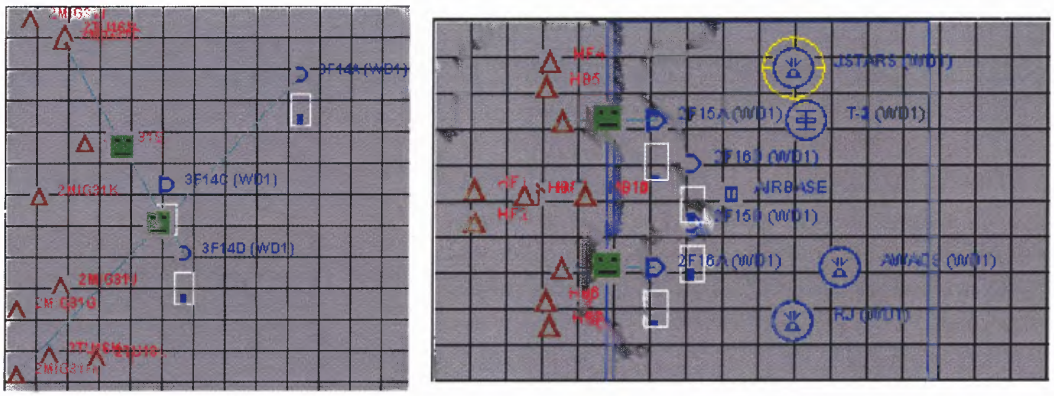

Figure 17. Two examples of recommendations for friendly fighter groups to engage the enemy. The human director can query the rationale, accept or ignore the recommendations.

Once an order is issued, either manually entered by a human director, or automatically generated by an assistant agent, the order is then propagated to the recipient resource via communication links. If, during the course of an order propagation a communication link fails, the order may not reach the recipient or a proper confirmation may not be obtained. In this case the order is not accepted and will not be executed due to communication failure. This process is visualized by colors and animation, as shown on Figure 18.

This example shows that a transient failure in the communication subsystem of an aircraft could result in reduced effectiveness and even in loss of strategic advantage. In other cases failures may not have such severe effects, however, in order to prevent the possibility of (even small and transient) failures snowballing into significant losses, one needs to anticipate and neutralize potentially critical failures. Our tool, in this case, can help train a team of Weapons Directors to recognize and eliminate failure related reduced performance. Furthermore, the trainer can trace back the cause-effect links from specifically inserted faults and a potential individual or team mistakes.

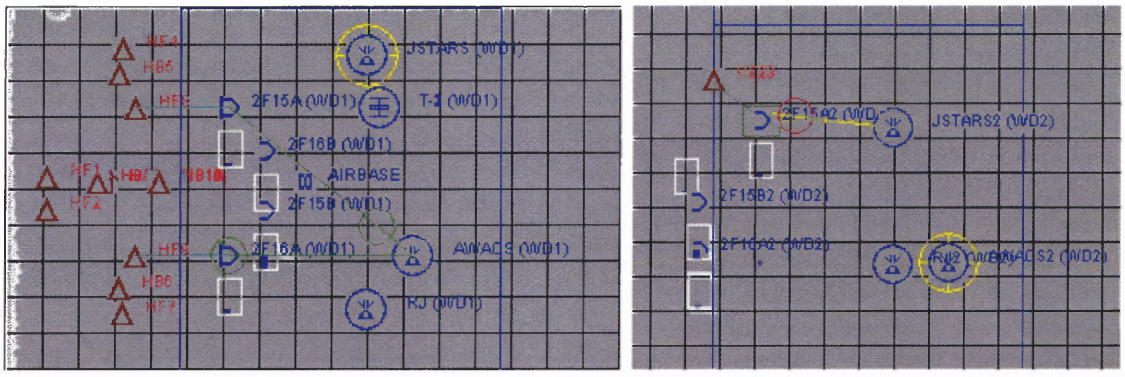

Figure 18. Order propagation visualization. Accepted recommendations caused two orders to be issued. In the left capture, the traveling green circles indicate normal communication of the orders. The returned red circle on the right indicates that the order has not been acknowledged by the fighter group.

The natural and visual GUI and function provided, makes it easy for a problem domain novice to use our system. In particular, a system engineer may examine how the designed simulated complex high integrity (in our case, we have a DoD application; other users may build others) behaves, and probe, through fault injection, how the design reacts to faults. Should the reaction(s) be excessive, an alternate design (resulting in a different

computer-communication-human-equipment architecture) will be tried, and, again, evaluated by injection more faults, and so on. The engineer will try until a suitable design is found.

Section 5.3 Agent Management and Interaction

5.3.1 Agents and Their Interactions

The decision support and training environment incorporates a variety of assistive agents. These are designed to monitor and collect information, maintain situational awareness, evaluate the current tactical picture and estimate mission effectiveness, evaluate current resource allocation and provide alternative allocations and so on. In this section we outline some of these agents and their interactions.

To classify the properties and responsibilities of agents, we use a description scheme consisting of the following categories: **type** of the agent, specific **characteristics**, **functions** the agent performs, **input** data for the agent, and **output** the agent produces. The agent type describes the generic class in which the agent belongs. Agents can belong to more than one class, based on their functionality, and to exactly one, based on their mode of operation. Examples for functionality based types are *allocator*, *evaluator*, *mediator*, etc. Agents can be classified as *active*, *passive* or *interactive*, based on their mode of operation. Active agents perform their tasks without being prompted, continuously and autonomously; in other words they execute their own thread of control.

Passive agents operate when triggered by events; the observe, listen and act only when there is information to be processed. Interactive agents are, for the most part, user-interface agents; they have their own thread of control, but it is designed around interacting with human users (and other agents). In the characteristics category we outline the features specific for the agent, its purpose and qualities – these are the features that make this type of agent unique in the hierarchy. The functions description for an agent is a more detailed and lower-level description of the agent's responsibilities. While the characteristics specify *what* the agent's purpose is, the functions describe *how* that purpose is accomplished. Finally, the input and output categories describe the agent's interactions with other agents and potentially with users. The input specifies what information the agent requires in order to execute and what would be the source of this information. The output category discusses the type and structure of the information resulting from the agent's execution. It also describes who the consumers of this information might be.

In Table 2 we present a set of possible agents and their properties in tabular form.

Table 2. Agent types and their properties

Facilitator agent	Type	Active, mediator, allocator
	Characteristics	Interfaces with other agents; adaptive to multiple interfaces
	Functions	Listens for advertised services and service requests. Matches (allocates) services to requests. Enforces rules of agent interaction.
	Input	Advertisement of services from providers; Requests for services; rules and primitives for request descriptions (ontology, agent communication language).
	Output	Dispatch of service requests to service providers
Sanity Checker agent	Type	Passive, evaluator
	Characteristics	Available on demand; re-configurable; self-adaptive to user patterns; interactive.
	Functions	Performs analysis and verification of the current platform-to-target pairings against a set of pre-defined objectives, values and rules of engagement. The agent will be invoked towards the end of each (re)targeting cycle, before orders are issued to the resources and made available to other commanders.
	Input	A set of orders, a set of rules for evaluation and goals.
	Output	Evaluation of the orders and comparison to previous orders.
User Guide agent	Type	Passive, librarian
	Characteristics	Available on demand; presentation oriented interfaces; tracks and updates current state information; may be mobile/distributed (on-demand mode).
	Functions	Employs information classifications, search, fusion and pattern matching to compose a context-sensitive user guidance information. Tracks the current state and live-updates the help as the situation requires.
	Input	User actions, user interface state.
	Output	Integrated, updated user guide information and documents.
Component Hunter agent	Type	Active, researcher
	Characteristics	Working in the background; mobile; autonomous; low-profile interactions; communicates with other agents
	Functions	Searches, identifies and evaluates available components. The developer specifies the interfaces, functional description, pattern or properties of the required components and releases the Hunter into the distributed design environment. The Hunter interacts with component brokers, component databases, possibly component developers (humans) and identifies potential matches for the required components. The hunter may accept performance constraints from the user, e.g. "Complete task within 30min." or "Quit after first 90% match."
	Input	Component descriptions (interfaces, functional and non-functional requirements), performance requirements
	Output	A set of matching components and match confidence estimates.

Advisor agent	Type	Active, evaluator, allocator
	Characteristics	Operates in the background; pops up in interactive mode; configurable, adaptive
	Functions	Observe user's command process and evaluate/compare the current set of orders/allocations against existing models. Identify and evaluate alternative allocations or orders to attempt to arrive at a better tactical situation.
	Input	Current orders and allocation (snoop mode), user activities (snoop mode), goals and objectives for the allocation, rules and functions for evaluation of the alternative orders.
	Output	Recommendations and rationale for better/alternative allocations.
Fault Monitor agent	Type	active, monitor, evaluator, actor
	Characteristics	Runs on behalf of the simulated entities; adaptive, autonomous, mobile if necessary
	Functions	Monitor the prototyped system for anomalies and malfunctions. In case a fault is detected, evaluates the situation and possibly requests help from mediators. Once the symptom and the cause of the malfunction are determined, specifies actions to be performed, including possible self-repair of transient faults.
	Input	Data collected from the simulated entities.
	Output	Recommendations for action, such as switching to backup, emergency mode, shutdown, or self-repair.
User interface agent	Type	Interactive.
	Characteristics	Responsive (near RT), user-oriented, adaptive to user requirements and to different user profiles
	Functions	Could be either input or output or both. Input agents listen to user input or are triggered by a user-generated events (mouse, keyboard...), then the collect, process and analyze the user input and determine appropriate action. Output agents are responsible for data presentation, graphical rendering, audio and video presentations. Speech recognition agents and speech synthesizers are an example.
	Input	User generated input, data from other agents
	Output	Rendering data in human-understandable form, requests to other agents

Table 2. Agent types and their properties

CHAPTER 6

EVALUATION OF THE PROTOTYPE

Section 6.1 Measuring the Agent Performance

The goal of the agent-based decision support system presented in this work is to enhance the performance of command and control personnel aboard the AWACS and to increase the effectiveness of their training with simulated scenarios. To ascertain the benefits of this decision support system we conducted two types of evaluations. First, we went through an iterative process of problem domain expert evaluation and corresponding enhancement in the prototype. Second, the system was evaluated with live WD subjects, using the agent decision support in a series of simulated training exercises. Both individual and team performance tests were conducted. The effectiveness of the agent recommendations was judged by the WD performance improvement when they used agent help versus their nominal performance when no help was available.

Domain expert evaluation. During the first stage of the evaluation process, military problem domain experts were used to evaluate the validity of the decision support system. Experts were drawn from the pool of active and retired Weapons Directors, Senior Directors and training personnel. These persons have learned, participated, observed and supervised the processes and operations modeled by agents in the decision support system. One of the roles of domain experts was to design scenario vignettes, which provided case-based testing and evaluation. Specific agent recommendations were

tested and fine-tuned in this manner. Examples include recommendations for in-air refueling, SAM avoidance navigational agents, high-threat detection and re-targeting etc. Another method used by the domain experts to test the system was through first-person evaluation, where the expert himself would participate in a training scenario and observe the agent behavior. In this setting, the expert is guided by agent system, though not required to accept its recommendations. Should a difference be detected between an agent recommendation and the expert's opinion, the simulated exercise would be paused the tactical situation analyzed. While this process is subjective, based on the expert's experience and intuition, it is well structured and repeatable. If adjustment in the agent behavior was required, the same scenario was tested again for the same behavior. This iterative process ensured that the operation of our heuristics and objective functions closely matched the expectations of experts. In a sense the process is similar to training a neural net with a set of pre-classified data. Unfortunately, in the domain of large complex command and control problems, such data sets are nearly impossible to construct, and the only opportunity for capturing domain expertise is through human-in-the loop exercises with problem domain experts.

In order to evaluate agent effectiveness based on the improvement of performance of trainee WDs, special load and performance measures were constructed. Based on the premise that if we can accurately measure the human's performance with and without the agent help, this would then indicate the value of the decision support agents. This is true, because in a single-subject simulated exercise, the only variable is the performance of the human WD. To account for the performance of the WD, three types of scores are

maintained in the simulation test-bed: (a) individual WD score, (b) team or SD score, and (c) mission score. The individual WD score is based on the ability of the WD to preserve friendly resources and to destroy attacking enemy resources. It is a composite score, based on the number of hostile kills, the number of aircraft lost, and other factors. The scoring is performed when any one of a set of scoring events occurs in the simulation. The score history, as well as the scoring events are collected for analysis. The team (or SD) score represents the combined scores of the members of the team of WDs under one Senior Director. It is a measure of how well the entire team of WDs performs in the exercise. The mission score is based on a composite performance measure, described by Dalrymple [MD96?]. This score includes the following attributes:

- (number of friendly assets not destroyed)^2 (added)
- number friendly assets destroyed (FACd) (subtracted)
- number hostile assets destroyed by friendly action (HACd) (added)
- kill ratio = sin(arctan(HACd/FACd)) * HACd (added)
- friendly air refuelings completed (added)
- assign/defer actions completed (transfer responsibility for resource) (added)
- number friendly lost to fuel out (subtracted)
- number friendly lost to friendly SAM missiles (subtracted)
- number friendly lost to friendly aircraft fire (subtracted)
- number hostile aircraft jammed (added)
- number friendly aircraft jammed (subtracted)
- maximum distance friendly aircraft penetrated hostile territory (added)

- hostile penetration of friendly area (subtracted)
- hostile aircraft lost to fuel out (added)

Individual Weapons Director scores are based on a ratio of handled threats to all potential threats to the particular director. Threats' values are based on the weapons strength and the range of the hostile entity. The following attributes are considered for individual scores:

- threat value of hostile aircraft entering the WD's area of responsibility
- thereat value of hostile resources destroyed
- number of in-air refueling operations completed
- number of resources lost due to fuel-outs
- value all of aircraft lost
- value of bases, SAM sits, or cities lost

Mission and individual scores are collected throughout the simulated exercise for all participating commanders, weather played by human WDs or by agent surrogates. When a score set is collected for a commander played by agents in a particular scenario, this establishes a base-line for the experiment. Next, as score set is collected for the same commander role, played by a human WD without decision support. Finally, the a score set is collected for a human WD with the decision support turned on. The procedure can be repeated with slightly modified scenarios or with sets of varying scenario difficulties, in order to establish under what conditions the decision support is most effective Also,

the three sets of scores (agents only, human with no decision support, human with decision support) can be collected from different subject groups. For example, experienced WDs may react differently to the agent-based decision support than novice or trainee WDs.

In addition to the performance measures, a host of other measures are collected during a simulated exercise. Load measures indicate the level of cognitive burden, and consequently stress and fatigue, caused by multiple incoming hostile threats. Load may be caused not only by shear numbers, but also by types of hostile threats. For example, the detection of a cruise missile, which may carry a strategic weapon is likely to dramatically increase the stress level of a defensive WD. Stress and situational awareness can be also measured by the reaction of the human WD to the information presented to him. Frequent display manipulations, such as zooming in and out and panning the display indicate high level of awareness, while frequent examination of resource properties without issuing an order may indicate high stress or risk aversion. Reducing the stress and fatigue, while increasing the situational awareness of the human commander can be accepted as an indication of success for the decision support system.

Section 6.2 Results

In this section we present the results of multiple evaluation sessions for the agent-based decision support environment. The data was collected both at 21st Century Systems, Inc. and at Brooks AFB. The experiments at Brooks AFB involved active and in-training

Weapons Directors, who used the environment in simulated exercises to validate the benefits of decision support to team command and control training and operations. In the experiments, the WD ran several scenarios both with and without the decision support agents enabled. When the exercise is run without agent support, the trainee WD is still able to view information about the resources she controls and limited information about the enemy resources, however no recommendations are displayed as to what action is to be taken. When the exercise is run with decision support agents enabled, the WD benefits from recommendations for actions, such as targeting assignments, refueling recommendations, SAM avoidance, and others. We emphasize this point, since the two modes – agents ON and agents OFF – differ only in the availability of decision support to the trainee. Independently, as confirmed by experts, that the visualization and distributed interactive simulation already provide significant benefit for command and control training. The purpose of the evaluation is to verify that the agent-based decision support system, which is the primary contribution of this work, significantly extend the benefits of distributed mission training and can bridge those advances to the operational environments (such as the AWACS or other command and control centers).

First we present the results of the experiments performed at Brooks AFB with actual WDs. During the limited time of availability of subjects large amounts of data were collected were collected owing to the automated load and performance measures implemented as part of the agent system. Here we will focus on the performance data, gathered from multiple runs of six different scenarios. All scenarios were based on an air defense of an island, attacked by enemy forces, followed by a counter-attack with

multiple strike missions. Even though the scenarios featured multiple WD roles, such as Defensive Counter-Air (DCA), Offensive Counter-Air (OCA), Suppression of Enemy Air Defenses (SEAD) and Strike, as well as control of Navy, Allied and Hostile forces, the scenarios were designed to train and test the capabilities of Airforce DCA controllers – thus we focus on the performance of the AF-DCA directors across all scenarios. The data we analyzed is based on the performance scores of individual directors and teams. We maintained workload the same across all scenarios in order to correlate the performance data with the availability of decision support or the lack thereof.

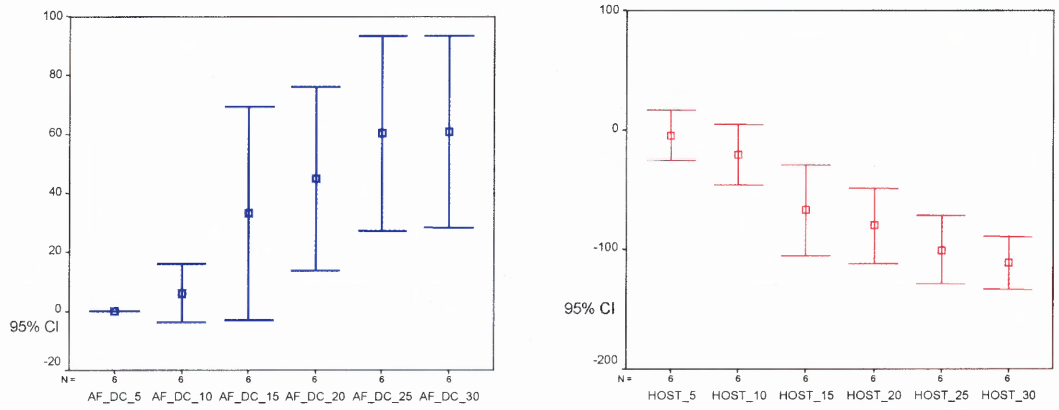

Figure 19. Average scores of AF-DCA and hostile DCA directors

On Figure 19 we present the average performance scores of a friendly and a hostile DCA directors. The friendly AF-DCA role was played by a human WD with agent decision support turned on. The hostile DCA was in automatic mode. The confidence intervals represent the variation in performance among the different scenario runs. We speculate that these are due to two factors: (1) the necessity of humans to get used to agent recommendations and to build trust in the software, and (2) the relatively small samples we used. We argue that for large samples of WDs who are beyond the learning curve of

the tool, the results will converge to the average values captured here. The difference between the scores of the friendly and hostile DCA directors is based primarily on the specific characteristics of the scenario. In this particular case the friendly forces dominated the hostile forces. These graphs represent a typical time-event capture of the domain dynamics. The entire exercise takes 30 minutes, from which the highest level of activity occurs between 10 and 25 minutes. These middle 15 minutes are critical for the air-battle and for achieving air-superiority. The period between the 5 and the 15 minute mark is therefore when most critical decisions for weapon-target pairing are made.

Let us now consider the performance of a human WD without agent decision support. On Figure 20 we see the performance of a friendly DCA director in the same 30 minute period with his advice agent turned off. . In this typical example we note that the unaided WD scores linger around a particular value – the equilibrium of forces based on the scenario and the skill of the trainee. While the scores improve in the initial period from 5 to 10 minutes, when the air-battle evolves and the workload increases, the unaided WD becomes inefficient and his scores drop.

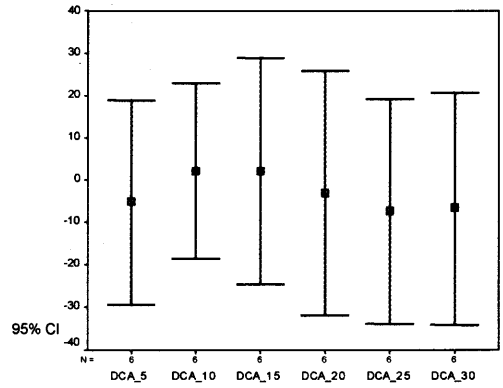

Figure 20. Performance of a DCA director without decision support

If we compare the first (friendly DCA) graph on Figure 19 with the performance of the unaided WD on Figure 20, we conclude that the agent based decision aid did provide an advantage. The benefit came from identifying critical decisions early on and consistently improving throughout the exercise. The worse performance of the unaided WD can usually be attributed to overload, loss of SA and an array of psychological factors, such as risk aversion under stress and time constraints. The unbiased and consistent advice from the decision support system aims to improve these shortcomings.

On Figure 21 we present a different view of the collected and averaged scores – in this case we compare the hostile score to the combined friendly score at any particular time. The colors of data points correspond to the different time periods. We see that while most exercises start with a slight advantage for the hostile forces, the balance quickly changes and the center of gravity gradually shifts toward the lower-right corner, representing advantage of the friendly forces over the hostile. Since this trend is scenario-dependent, we can use the relationship in Figure 21 to disambiguate the scenario bias present in Figure 19.

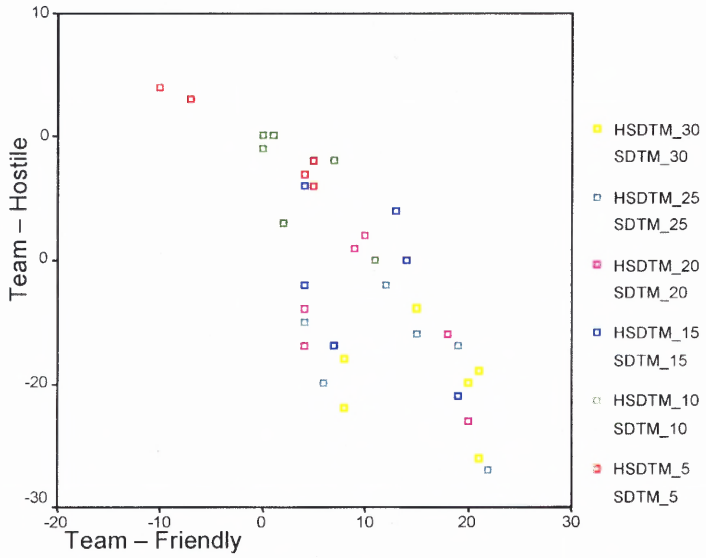

Figure 21. Comparative performance of friendly vs. hostile directors

The performance measures collected show clearly that under conditions of high workload, WDs with agent support performed consistently better than those without agent-based decision support. Based on these results, we claim that the agent-based decision support improves the performance of commanders in three major ways:

- Decisions are reached faster
- The decision-making process is more systematic
- The overall outcome is improved

In an environment where critical decisions have to be made under time pressure and overwhelming amounts of information, even simple automation and sub-optimal heuristic recommendations help the WD focus, organize information better and ultimately perform better.

CHAPTER 7

CONCLUSION

In this chapter we would like to summarize the results of the work and to discuss the possibilities for future extensions. Throughout the work we have attempted to associate and discuss research issues in the context of specific applications. In our view the application of agent-based decision support tools to the AWACS domain is not only beneficial in traditional sense, but also as an example of integration of two major directions in computing, driven by the needs of customers. We would like to continue researching, improving, and revolutionizing this area without departing from the practicality and applicability of this work, as many questions still remain unanswered.

Section 7.1 Results of the Work

The major contributions of this work can be classified in two major areas: a) multi-agent architectures with performance requirements, and b) simulation and agent systems integration. We have designed and implemented a multi-agent decision support tool, which is highly flexible and provides QoS guarantees, required by the nature of the decision support problem. The implementation includes a simulation test bed for the AWACS problem domain, an agent manager and six classes of agents: 1) data monitors and filter agents, 2) analysis, allocation and decision evaluation agents, 3) routing agents, 4) recommendation agents, 5) performance tracking and evaluation agents, and 6) tutorial

agents. The design was tested and evaluated with a number of training scenarios, designed with the help of AWACS domain experts, with part of the evaluation taking place at Brooks AFB.

Now we will revisit several key components of the system, which in our opinion best define the strengths and the emphasis of the work.

Situational awareness agents. Part of the monitoring and filtering class of agents, these collect, classify, and structure data in order to derive knowledge about the evolving tactical situation. These agents utilize rule-based logic, heuristics and multi-objective function evaluation to determine what information is important to the decision-making agents and humans. These agents are capable of dynamically adapting to changing priorities in the decision making process and to a very dynamic environment of information sources. As filters and organizers, these are the first, very important step in the reduction of problem space to a manageable size.

Resource allocation agents. These agents belong to the analysis and allocation class, which is the heart of the decision support system. The main function of the resource allocators is to efficiently apply rules and heuristics and to derive alternative allocations. Objective functions are optimized when resource allocations are evaluated. Since each agent is responsible for a subset of all available resources, both local (within the team) and global (among all teams) objective optimization needs to be performed. Thus, resource allocators need to communicate with each other and with other agents. A strict

bound on the time and computing resources needed to evaluate allocations is maintained and updated, so that the resource allocators are able to comply with their QoS and timing requirements.

Agent manager with QoS support. The agent manager servers two roles in our model – it coordinates all agent activity, including inter-agent communications, dispute arbitration, computing resource allocation and performance monitoring etc., and it also serves as an interface to the simulation engine. A careful multithreaded design was required in order to avoid bottlenecks at the agent manager. Communications via shared memory structures and over remote object interfaces is facilitated by the agent manager to keep local and remote agents synchronized and to allow for simulation data access.

Multi-agent architecture. The amount of domain specific rules and requirements for the agent-based decision support system was the basis of our decision to develop a new multi-agent architecture. The need for flexible, modular system with the possibilities of extensions along multiple functional axes, as well as the need for adaptive, autonomous behavior of decision aids determined our choice of an agent-based environment. The requirements for performance and QoS guarantees, as well as the need for integration with a time-event simulation guided us towards the architecture we developed and presented here.

Decision support and situational awareness. The overall goal of this work has been to design, develop and test advanced, novel techniques for command and control decision

support. Decision making in time-critical, high-stress, information overloaded environments, such as the AWACS domain, is a complex, demanding task. The need for operational automation and for better training becomes apparent, unfortunately through numerous incidents, which could have been prevented, had a better technology been available to the overloaded personnel. Although it is a hard problem to address, we feel we have reached reasonable success with our decision aid tools to improve the situational awareness and provide alternatives and recommendations in order to reduce the operator's work load. Better situational awareness is achieved by automating monitoring tasks, such as fuel and munitions tracking, and allowing the human decision-makers to focus on the tactical picture. When an extraordinary condition is detected, the operator's focus is immediately directed to it (for example, detection of new threat, or request for support by allies), alternatives are presented and evaluated with respect to the current mission objectives. The technology, thus, aims to support the human operator, rather than to replace him.

DSS tool evaluation. Since the developed decision support tool is aimed at improving the performance of human WDs (and other decision-makers), the only true way to evaluate it's effectiveness is to measure the change of effectiveness of WDs who use the tool against those who do not. There are other, supporting measures of the capabilities of the tool and the authenticity of the underlying models, though we value the most the opinions of the ultimate users. During actual evaluations by USAF personnel, active and retired WDs have noted the effectiveness of the environment for tactical training. The ability to quickly create and evaluate scenarios, enabled by an online visual scenario

editor and a automatic agent-only mode of execution, has proven very valuable to training teams. While the general impression was that the DSS technology helps alleviate the workload of WDs, it was also interesting to note that some WDs did not fully trust the agent's recommendations because they did not understand intuitively how the recommendations were derived. This said, the overwhelming need for decision aiding tools in the command and control area far outweighs the initial drawbacks associated with the learning curve for the technology.

Section 7.2 Future Work

The research, implementation and interactions with the domain experts, associated with this work, have proven to be challenging and, at the same time satisfying. While the prototype has reached a fairly stable state, there are a number of new ideas that are intriguing to us and, we speculate, of interests to the community. The work has several natural venues for expansion: 1) extensions of the model to include uncertainty of information and probabilistic behaviors, 2) extensions to the decision-maker hierarchy (and modeled physical entities) to support a range of decision-makers at multiple levels of the command chain, 3) extensions of the DSS tool to applications in planning, routing, real-time re-routing and re-targeting. Preliminary work on some of these items has began as of this writing, and others are a very near future possibilities. In the long term, the model can be extended to include very interesting, though quite computationally intensive applications of dynamic game theory, stochastic modeling of adversarial

behavior. The work is, without a doubt, applicable to an array or related fields, from air-traffic control, to financial market predictions, to management of large organizations.

The future of the agent-based decision support system seems promising. The technology can be applied not only to the AWACS problem, but to all military command and control domains, where time and mission critical decisions have to be made under conditions of stress and overwhelming amounts of information. As the system is tested and used in different settings, additional venues for improvement will no doubt be uncovered.

APPENDIX A. TERMS AND ABBREVIATIONS

ATO	Air Tasking Order
AWACS	Airborne Warning and Control System
C4I	Command, Control, Communications, Computers and Intelligence
CAP	Combat Air Patrol
Comm	Communication
DIS	Distributed Interactive Simulation
DSS	Decision Support System
GUI	Graphical User Interface
(H)TLAM	(Hostile) Tactical Land-Attack Missile
JIT	Just-in-time (Java compilers)
NCO	Non-commissioned officer
ORB	Object Request Broker
RMI	Remote Method Invocation (in Java)
ROE	Rules of Engagement
RT	Real-time
RTB	Return to Base
SA	Situational Awareness
SEAD	Suppression of Enemy Air Defenses
SD	Senior Director
Sim	Simulator (or Simulation)
UAV/UCAV	Unmanned Aerial (Combat) Vehicle
VUI	Visual User Interface
WD	Weapons Director

BIBLIOGRAPHY

1. **[AA95]** Andriole, S., & Adelman, L. (1995). *Cognitive Systems Engineering for User-computer Interface Design, Prototyping, and Evaluation*. New York: Lawrence Erlbaum.

2. **[ABMS98]** C. Amaro, S. Baruah, T. J. Marlowe, A. D. Stoyen, "Nonpreemptive Scheduling to Maximize the Minimum Intercompletion Time," to appear *in Journal of Combinatorial Mathematics & Combinatorial Computing*, 1998.

3. **[AMS96]** C. Amaro, T. J. Marlowe, A. D. Stoyenko, "An Approach to Constructing Complex Evolving Systems Using Composition of Knowledge Domains," *21st IFAC/IFIP Workshop on Real-Time Programming*, November 1996.

4. **[Amdahl98]** Amdahl Product Descriptions. Available online at http://www.amdahl.com/doc/products 1998.

5. **[AFJM95]** R. Armstrong, D. Freitag, T. Joachims, and T. Michell, "WebWatcher: A Learning Apprentice for the World Wide Web," in *Proceedings of AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments, 1995.*

6. **[Betalii95]** Bhatt, Devesh, Rakesh Jha, Todd Steeves, Rashmi Bhatt, and David Wills, "SPI: An Instrumentation Development Environment for Parallel/Distributed Systems," Proc. of Int. Parallel Processing Symposium, April 1995.

7. **[BB98]** Bigus, J.P., & Bigus, J. (1998). *Constructing Intelligent Agents with Java*. New York: John Wiley & Sons.

8. **[BG88]** A.Bond, and L. Gasser, "Reading in Distributed Artificial Intelligence": Morgan Kaufman, New York, NY, 1988

9. **[BZW98]** W. Brenner, R. Zarnekow, & H. Wittig. *Intelligent Software Agents*. Amsterdam: Springer-Verlag. 1998

10. **[BHMM94]** Brown D., S. Hackstadt, A. Malony, B. Mohr, "Program Analysis Environments for Parallel Language Systems: The TAU Environment," Proc. of the 2nd Workshop on Environments and Tools For Parallel Scientific Computing, pp. 162–171, May 1994.

11. **[CMS99]** S. K. Card, J. D. Mackinlay, & B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. New York: Morgan Kaufman. 1999.

12. **[CH97]** A. Caglayan, & C. Harrison. *Agent Sourcebook*. New York: Addision Wesley. 1997.

13. **[CKPW97]** A. Castillo, M. Kawaguchi, N. Paciorek, D. Wong. Concordia as Enabling Technology for Cooperative Information Gathering. 1997.

14. **[Cetalii95]** M. Calzarossa et al., ``Medea: A Tool for Workload Characterization for Parallel Systems," *IEEE PDT*, V3, N4, Winter 1995.

15. **[CMS83]** S. Cammarata, D. McArthur, and R. Steeb. "Strategies of Cooperation in Distributed Problem Solving," IJCAI-83.

16. **[CS97]** C. Cheng and S.F. Smith. "Applying Constraint Satisfaction Techniques to Job Shop Scheduling," Annals of Operations Research, Special Issue on Scheduling: Theory & Practice, 70:327-357, 1997.

17. **[CS98]** L. Chen, and K. Sycara. "WebMate: A Personal Agent for Browsing and Searching," In Proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, Minneapolis, MN, May 10-13, 1998.

18. **[CSS91]** Cognitive Science Society, Chicago, Ill., 1991, Pages 400-405.

19. **[C78]** P.R. Cohen, "On Knowing What to Say: Planning Speech Acts," Ph.D. Thesis, Department of CS, University of Toronto, 1978

20. **[D87]** E.H. Durfee. Ph.D. Thesis (Durfee.drt): "A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network," University of Massachusetts, Amherst, MA.1987

21. **[Detalii92]** L. Dent, J. Boticario, J. McDermott, T. Michell, and D. Zabowski. "A Personal Learning Apprentice," Proceedings of AAAI92

22. **[DLC85]** E. Durfee, V. Lesser, and D. Corkill. "Coherent Cooperation Among Communicating Problem Solvers," Department of Computer Science and Information Science, University of Massachusetts, Amherst, MA

23. **[DPSW97]** K. Decker, A. Pannu, K. Sycara, and M. Williamson "Designing Behaviors for Information Agents," In Proceedings of the First International Conference on Autonomous Agents, Los Angeles, February 1997.

24. **[DRSL98]** Dependable Real-Time Systems Lab at NJIT, "A Hierarchy and General Equational Form for Cost and Objective Functions for Complex Real-Time Systems," *an ongoing report*, 1993-1998.

25. **[DS83]** R. Davis and R.G. Smith. "Negotiation as a Metaphor for Distributed Problem Solving," Artificial Intelligence, 20:63--100, 1983

26. **[DS97]** K. Decker and K. Sycara. "Intelligent Adaptive Information Agents." Journal of Intelligent Information Systems, 9:239--260, 1997.

27. **[DSW97]** K. Decker, K. Sycara, and M. Williamson. "Middle-Agents for the Internet." In Proceedings of IJCAI-97, 1997

28. **[DSW96]** K. Decker, K. Sycara, and M. Williamson, "Intelligent Adaptive Information Agents".

29. **[AAAI96]** In Proceedings of the AAAI-96 Workshop on Intelligent Adaptive Agents, Portland, Oregon, August 4-8, 1996.

30. **[DWS96]** K. Decker, M. Williamson, and K. Sycara. "Matchmaking and Brokering," In Proceedings of the Second International Conference on Multiagent Systems, 1996.

31. **[EF96]** S. Eick, D. Fyock, "Visualizing Corporate Data," AT&T Technical Journal, 75(1), Jan./Feb. 1996, pp. 74-86.

32. **[ES99]** L. R. Elliott, & S. Schiflett. "Development of Synthetic Team Training Environments: Application to USAF Command and Control Aircrews," To appear in H. O'Neil & D. Andrews (Eds.) *Aircrew Training: Methods, Technologies, and Assessment.* 1999.

33. **[EW94]** O. Etzioni and D. Weld. "A Softbot-based Interface to the Internet," Communications of the ACM, jul 1994, v.37, n.7

34. **[FRDD00]** R. P. Fahey, A. Rowe, K. Dunlap, and D. DeBoom.(in review). *Synthetic Task Design (1): Preliminary Cognitive Task Analysis of AWACS Weapons Director Teams.* Technical Report. Brooks AFB, TX: Armstrong Laboratory. 2000.

35. **[F95]** T. Fahringer, "Estimating and Optimizing Performance for Parallel Programs," *IEEE Computer*, Vo. 28, No. 11, November 95.

36. **[FFMM94]** T. Finin and R. Fritzson and D. McKay and R.McEntire. "KQML as an Agent Communication Language", Proc. of the Third International Conference on Information and Knowledge Management CIKM'94, ACM Press, Nov. 1994,

37. **[FMY92]** R. Farrow, T. J. Marlowe, D. M. Yellin, "Composable Attribute Grammars: Support for Modularity in Translator Design," 223-234, *ACM 1992 Principles of Programming Languages,* January 1992.

38. **[GK94]** M. R. Genesereth and S. P. Katchpel. "Software Agents", Communications of the ACM, 1994, v 37, n 7, p48--53,147

39. **[GS92]** P.A. Gibson, A. D. Stoyenko, "Development and Integration of a Concurrently Executing Interactive User Interface for the I-STAT Portable Clinical Analyzer: A Case Study in Real-Time Systems Integration," *J. Sys. Integration*, 2 (4), 349 - 388, Oct. 1992.

40. **[Getalii94]** Gu, Weiming, Greg Eisenhauer, Eileen Kraemer, Karsten Schwan, John Stasko, and Jeffrey Vetter, "Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs," Technical Report GIT–CC–94–21, 1994

41. **[HKWJ95]** Hao, Ming C., Alan H. Karp, Abdul Waheed, and Mehdi Jazayeri, "VIZIR: An Integrated Environment for Distributed Program Visualization," Proc. of Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '95) Tools Fair, Durham, North Carolina, Jan. 1995.

42. **[H96]** J. Hendler. Intelligent agents: Where AI meets information technology. *IEEE Expert*. 1996.

43. **[HS91]** W. A. Halang, A. D. Stoyenko, "Constructing Predictable Real-Time Systems," *Kluwer Academic Publishers*, 1991

44. **[HMSS95]** M. S. Harelick, T. J. Marlowe, A. D. Stoyenko, P. Sinha, "A Constraint Function Classification for Complex Systems Development," *1995 Complex Systems Engineering and Assessment Technology Workshop*, Ft. Lauderdale, FL, November 1995.

45. **[HSR93]** **A.** Hayes, M. Simmons, and D. Reed, "Workshop Report: Instrumentation for Parallel Computer Systems: A Dialogue Between Users and Developers," Keystone, Colorado, April 1993.

46. **[AT&T98]** T. He and S. G. Eick. "Constructing Interactive Network Visual Interfaces," Bell Labs Technical Journal, available online at http://www.lucent.com/minds/techjournal/apr-jun1998/pdf/paper04.pdf 1998.

47. **[HE91]** M. T. Heath, and J. A. Etheridge, "Visualizing the Performance of Parallel Programs," IEEE Software, 8(5), September 1991, pp. 29–39.

48. **[HMR95a]** M. T. Heath, A. Malony, and D. Rover, "The Visual Display of Parallel Performance Data," *IEEE Computer*, 28(11), November 1995, pp. 21-28. Special issue on Performance Evaluation Tools for Parallel and Distributed Systems.

49. **[HMR95b]** M. T. Heath, A. Malony, and D. Rover, "Parallel Performance Visualization: From Practice to Theory," *IEEE Parallel and Distributed Technology*, 3(4), Winter 1995, pp. 44-60. Special issue on Performance Evaluation Tools for Parallel and Distributed Systems.

50. **[HMR96]** M. T. Heath, A. Malony, and D. Rover, "Visualization for Parallel Performance Evaluation and Optimization," in *Software Visualization: Programming as a Multimedia Experience*, edited by M. Brown, J. Domingue, B. Price, and J. Stasko, MIT Press, 1996.

51. **[H96]** J. Hendler. Intelligent agents: Where AI meets information technology. *IEEE Expert*. 1996.

52. **[H91]** N. D. Hoang, "The Essential Views of Systems Development," *Proceedings of 1991 Systems Design Synthesis Technology Workshop,* Naval Surface Warfare Center, Silver Spring, Maryland, 3-9, September 1991.

53. **[HM93]** Hollingsworth, J. K. and B. P. Miller, "Dynamic Control of Performance Monitoring on Large Scale Parallel Systems," Proc. of Int. Con. on Supercomputing, Tokyo, Japan, July 19–23, 1993.

54. **[Jetalii93]** D. Jablonowski et al. "VASE: The Visualization and Application Steering Environment," *Proc. Supercomputing'93*, pp. 560-569.

55. **[JSW98]** N. Jennings, K. Sycara, and M. Wooldridge. "A roadmap for agent research and development," Autonomous Agents and Multi-Agent Systems, 1(1), 1998.

56. **[KKTW96]** G. L. Kaempf, G. Klein, M. L. Thordsen, S. Wolf, "Decision Making in Complex Naval Command-and-Control Environments," *Human Factors*, 38(2), pp. 220-231. 1996.

57. **[KS86]** E. Kligerman, A. D. Stoyenko, "Real-Time Euclid: A Language for Reliable Real-Time Systems," *IEEE Transactions on Software Engineering*, Vol. 12, No. 9, pp. 941 - 949, September 1986.

58. **[Ketalii94]** Kleinfeldt, S. et al., "Design Methodology Management," Proc. IEEE, 82(2), 1994, pp. 231–250

59. **[Ketalii98]** E. Kraemer, et al., "Balancing Consistency and Lag in Transaction-Based Computational Steering," *HICSS-31*, January 1998.

60. **[K97]** S. Kraus, "Negotiation and cooperation in multi-agent environments," *Artificial Intelligence*, 94, pp. 79-97. 1997.

61. **[KH95]** D. Kuokka and L. Harada. "On using KQML for matchmaking," Proceedings of the First International Conference on Multi-Agent Systems, pages 239--245, San Francisco, June AAAI Press. 1995.

62. **[KHBM96]** S. D. Kushner, C. H. Heithecker, J. A. Ballas, D. C. McFarlane, "Situation Assessment Through Collaborative Human-Computer Interaction," *Naval Engineers Journal*, 108(4), pp. 41-52, July 1996.

63. **[LC83]** V. Lesser and D. Corkill, "The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks." The AI Magazine,V4,N3, P.15-33, Fall 1983

64. **[LE-R93]** T. Lewis and H. El-Rewini, "Parallax: A Tool for Parallel Program Scheduling," *IEEE Parallel and Distributed Technology*, Vol. 1, No. 2, pp. 62-72, May 1993.

65. **[LL89]** S. Lander and V. Lesser. "A Framework for Cooperative Problem-Solving Among Knowledge-Based Systems", Proceedings of the MIT-JSME Workshop on Cooperative Product Development, Cambridge, MA 1989

66. **[LO98]** D. B. Lange, & M. Oshima. (1998). *Programming and Deploying Java Mobile Agents with Aglets*. Reading, MA: Addison Wesley.

67. **[LMM99]** Y. Lashkari, M. Metral, & P. Maes. "Collaborative interface agents," In M. Huhns, & M.P. Singh (Eds.), *Readings in Agents*. San Francisco, CA: Morgan Kaufmann, 1999.

68. **[LMS96]** P. A. Laplante, T. J. Marlowe, A. D. Stoyenko, "Language Mechanisms for Real-Time Image Processing," *Control Engineering Practice*, 1996.

69. **[LS93]** J. Liu and K. Sycara. "Collective Problem Solving Through Coordinated Reaction," In Proceedings of the IEEE International Conference on Evolutionary Computation, pages 575--578, 1993.

70. **[lS95a]** J. Liu and K. Sycara. "Exploiting Problem Structure for Distributed Constraint Optimization," In Proceedings of the First International Conference on Mutli-Agent Systems, pages 246--253, 1995.

71. **[LS95b]** J. Liu and K. Sycara. "Emergent Constraint Satisfaction Through Multi-agent Coordinated Interaction." In C. Castelfranchi and J.-P. Muller, eds, From Reaction to Cognition, V957 of Lecture Notes in AI, Subseries of LNCS, pages 107--121. Springer-Verlag, 1995.

72. **[ICMAS95]** Proceedings of the First International Conference on Multi-Agent Systems (ICMAS), San Fransisco, CA., 1995

73. **[LS99]** A. Lux, & D. Steiner. "Understanding Cooperation: An Agent's Perspective," In M. Huhns, & M.P. Singh (Eds.), *Readings in Agents.* San Francisco, CA: Morgan Kaufmann. 1999.

74. **[M94a]** P. Maes, "Agents that Reduce Work and Information Overload," Communications of the ACM, V37, N7, P.31-40, 1994

75. **[MSYetalii98]** MacMillan, J, Serfaty, D., Young, P., Klinger, D., Thordsen, M., Cohen, M., & Freeman, J. *A system to enhance team decision making performance: Phase 1 Final Report.* Brooks AFB: Air Force Research Laboratory, Warfighter Training Research Division. 1998.

76. **[M94b]** T. J. Marlowe, "Flow-Sensitivity and Incremental Algorithms for Data Flow Analysis," *Workshop/Seminar on Incremental Algorithms*, Schloss Dagstuhl, Germany, May 1994 (invited).

77. **[MBHM94]** A. Malony, D. Brown, S. Hackstadt, and B. Mohr, "Program Analysis Environments for Parallel Language Systems: The TAU Environment", Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing, May, 1994, pp. 162-171.

78. **[Metalii93]** Malony, A., B. Mohr, P. Beckman, D. Gannon, S. Yang, F. Bodin, and S. Kesavan, "Implementing a Parallel C++ Runtime System for Scalable Parallel Systems," Proceedings of Supercomputing '93, Portland, Oregon, November 15–19, 1993.

79. **[MR90]** T. J. Marlowe and B. G. Ryder, "An Efficient Hybrid Algorithm for Incremental Data Flow Analysis," *17th Annual ACM Symposium on the Principles of Programming Languages,* 184-196, January 1990.

80. **[MSMW94]** T. J. Marlowe, A. D. Stoyenko, S. P. Masticola and L. R. Welch, "Schedulability-Analyzable Exception Handling for Fault-Tolerant Real-Time Languages," *Real-Time Systems*, Vol. 7, No. 2, pp. 183-212, 1994.

81. **[Metalii96]** T. J. Marlowe, A. D. Stoyenko, P. Laplante, R. S. Daita, C. C. Amaro, C. M. Nguyen, S. L. Howell, "Multi-Goal Objective Functions for Optimization of Task Assignment," *Control Engineering Practice*, 1996.

82. **[MPI]** "MPI-2: Extensions to the Message-Passing Interface," *MPI Forum*, 1996-1997.

83. **[MR91]** T. Marlowe, B. Ryder, "Properties of Data Flow Frameworks: A Unified Model," *Acta Informatica*, 28(2), 121-164, 1991.

84. **[Metalii93]** T. J. Marlowe, A. D. Stoyenko, L. R. Welch, P. Laplante, S. P. Masticola, "Incremental Analysis for Reuse and Change in a Software Development Environment for Hard-Real-Time Systems," *IEEE RTOS*, May 1993.

85. **[MCM98]** D. L. Martin, A. J. Cheyer, D. B. Morgan. The Open Agent Architecture: A Framework for Building Distributed Software Systems. 1998

86. **[MMR95]** S. P. Masticola, T. J. Marlowe, B. G. Ryder, "Multisource Data Flow Problems," *ACM Transactions on Programming Languages and Systems*, (5), 777-803, September 1995.

87. **[MS96]** Microsoft. Distributed Component Object Model Protocol – DCOM/1.0. Available online at http://www.microsoft.com/activex/+DCOM 1996.

88. **[Metalii95]** B. Miller, et al., ``The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, Vo. 28, No. 11, November 1995.

89. **[NRC97]** National Research Council (1997). *Tactical Display for Soldiers: Human Factors Considerations*. Washington, D.C.: National Academy Press.

90. **[NB94]** Nutt, Gary J. and Clive F. Baillie, "Integrated Debugging and Tuning Environments," University of Colorado, October 1994.

91. **[NG95]** Nutt, Gary J. and Adam J. Griff, "Extensible Parallel Program Performance Visualization," Proc. of Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '95), Durham, North Carolina, Jan. 1995.

92. **[OMG97]** Object Management Group (OMG). The complete CORBA/IIOP 2.1 specification. Available online at http://www.omg.org/corba/corbiiop.htm 1997.

93. **[Petalii97]** S. Parker, et al., "An Integrated Problem Solving Environment: The SCIRun Computational Steering System."

94. **[PLJI86]** H. Parunak, P. Lozo, R. Judd, B. Irish. "A Distributed Heuristic Strategy for Material Transportation"

95. **[PS97]** P. V. Petrov, A. D. Stoyen, "Compiler Support for Non-intrusive Monitoring and Debugging of Real-Time Systems in the CRL Environment," 1997 IEEE Real-Time Systems Symposium, San Francisco, California, USA, December 1997.

96. **[CISM86]** Proceedings 1986 Conference on Intelligent Systems and Machines, Rochester, Michigan, 1986

97. **[RG91]** A. Rao and M. Georgeff. "Modeling rational agents within a {BDI}-architecture." Proceedings of Knowledge Representation and Reasoning, p. 473--484, 1991.

98. **[RG92]** S. Rao and M. P. Georgeff. "An abstract architecture for rational agents." Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR '92), p. 439-449, Cambridge, Massachusetts, October 23-29, 1992.

99. **[Retalii93]** B. Ries, R. Anderson, D. Breazeal, K. Callaghan, E. Richards, and W. Smith, "The Paragon Performance Monitoring Environment," Proceedings of Supercomputing '93, Portland, Oregon, Nov. 15–19, 1993.

100. **[Retalii92]** Reed, Daniel A., Ruth A. Aydt, Tara M. Madhyastha, Roger J. Noe, Keith Shields, Bradley W. Schwartz, "The Pablo Performance Analysis Environment," Dept. of Comp. Sci., Univ. of Ill., 1992.

101. **[Retalii96]** D. Reed, et al., ``The Next Frontier: Interactive and Closed Loop Performance Steering," *Proc. 25th ICPP*, 1996.

102. **[R94]** D. T. Rover, "Performance Evaluation: Integrating Techniques and Tools into Environments and Frameworks," Roundtable, Supercomputing '94, Washington DC, November 1994.

103. **[RWMB98]** D. Rover, A. Waheed, M. Mutka, and A. Bakic, ``The Application of Software Tools to Complex Distributed Systems: An Overview," *IEEE Concurrency* (to appear).

104. **[RZ94]** J. Rosenschein and G. Zlotkin. "Rules of Encounter", MIT Press, Cambridge, Mass.1994

105. **[RGK99]** D. Rus, R. Gray, & D. Kotz. Transportable Information Agents. In M. Huhns, & M.P. Singh (Eds.), *Readings in Agents*. San Francisco, CA: Morgan Kaufmann. 1999.

106. **[S93]** T. Sandholm. "An Implementation of the Contract net Protocol Based on Marginal Costs." AAAI'93, pp. 252-262, Washington, DC.

107. **[SD93]** S. Sen & E. Durfee. "The Effects of Search Bias on Flexibility in Distributed Scheduling," 12th Int. Workshop on DAI. 1993.

108. **[SSJ97]** O. Shehory, K. Sycara, and S. Jha. "Multi-agent Coordination Through Coalition Formation," In Intelligent Agents IV: Agent Theories, Architectures and Languages, M. Singh, A. Rao and M. Wooldridge, (Eds), Springer, Lecture Notes in Artificial Intelligence No. 1365, 1997, pp. 143-154.

109. **[SSSM99]** O. Shehory, K. Sycara, G. Sukthankar, and V. Mukherjee. "Agent Aided Aircraft Maintenance," In Proceeding of Agents-99, Seattle, WA., 1999.

110. **[S91b]** Y. Shoham. "AGENT0: A Simple Agent Language and its Interpreter," AAAI91, p. 704-709 July, 1991, Anaheim, CA.

111. **[SIS]** O. Shehory, S. Jha, and K.. Sycara. "Multi-agent Coordination Through Coalition Formation," Proc. ATAL-97, Providence, 1997.

112. **[SM96]** A. Silberman, Thomas J. Marlowe, "A Task Graph Model for Design and Implementation of Real-Time Systems," *Second IEEE International Conference on Engineering of Complex Computer Systems*, Montreal, Canada, October 1996.

113. **[SS93]** S. Smith and K. Sycara. "Flexible Coordination in Resource-Constrained Domains." Robotics Institute Technical Report CMU-RI-TR-95-02, December, 1993.

114. **[SK93]** J. Stasko and E. Kraemer, "A Methodology for Building Application-Specific Visualizations of Parallel Programs," Journal of Parallel and Distributed Computing, 18, 2, June, 1993, pp. 258-264.

115. **[SY86]** R. Strom, S. Yemini, "Typestate: A Programming Language Concept for Enhancing Software Reliability," *IEEE Trans. on Software Engin.*, Jan. 86, vol 12, pp. 157-171.

116. **[S87a]** A. D. Stoyenko, "A Real-Time Language with A Schedulability Analyzer," *Ph.D. Dissertation*, Dept.of CS, U. Toronto, 1987.

117. **[S87b]** A. D. Stoyenko, "A Schedulability Analyzer for Real-Time Euclid," *IEEE 1987 Real-Time Systems Symposium*, December 1987.

118. **[S91]** A. D. Stoyenko, "General Model and Mechanisms for Heterogeneous Model-Level RPC Interoperability," *IEEE 1991 Symposium on Parallel and Distributed Processing* , Dallas, Texas, USA, pp. 668 - 675, December 1991.

119. **[S94]** A. D. Stoyenko, "SUPRA-RPC: *SUB*program *PaRA*meters in *R*emote *P*rocedure *C*alls," *Software—Practice and Experience*, Vol. 24, No. 1, pp. 27 - 49, January 1994. Earlier version in the *IEEE SPDP'90*.

120. **[SML96]** A. D. Stoyenko, T. J. Marlowe and P. A. Laplante, "A Description Language for Engineering of Complex Real-Time Systems," *J. Real-Time Systems*, 1996.

121. **[SMY95]** A. D. Stoyenko, T. J. Marlowe, M. F. Younis, "A Language for Complex Real-Time Systems," *Computer Journal*, 38(5), 1995.

122. **[SL95]** A. D. Stoyenko, P.A. Laplante (editors) "Engineering of Complex Computer Systems: Fundamentals, Techniques & Applications," *IEEE Press*, since 1995.

123. **[SLHM94]** A. D. Stoyenko, P. A. Laplante, R. Harrison and T. J. Marlowe, "Engineering of Complex Systems: A Case for Dual Use and Defense Technology Conversion," *IEEE Spectrum*, Vol. 31, No. 11, pp. 32-39, December 1994.

124. **[SB94]** A. Stoyenko, T. Baker, "Real-Time Schedulability-Analyzable Mechanisms in Ada9X," *Proceedings of the IEEE*, Jan. 1994.

125. **[Setalii93]** A. D. Stoyenko, L. R. Welch, P. A. Laplante, T. J. Marlowe, C. Amaro, B.-C. Cheng, A. K. Ganesh, M. Harelick, X. Jin, M. Younis, and G. Yu, "A Platform for Complex Real-Time Applications," *1993 Complex Systems Engineering and Assessment Technology Workshop*, Beltsville, Maryland, USA, July 1993.

126. **[SH93]** A. D. Stoyenko, W. A. Halang, "High-Integrity PEARL: A Language for Industrial Real-Time Applications," *IEEE Software*, Vol. 10, No. 4, pp. 65-74, July 1993.

127. **[SMHY93]** A. D. Stoyenko, T. J. Marlowe, W. A. Halang, M. Younis, "Enabling Efficient Schedulability Analysis through Conditional Linking and Program Transformations," *Control Engineering Practice*, Vol. 1, No. 1, pp. 85-105, January 1993.

128. **[SMCG96]** A. D. Stoyenko, T. J. Marlowe, B.-C. Cheng, A. Ganesh, "Performance Prediction Functions for Real-Time Software Components," in consideration for *IEEE Transactions on Parallel and Distributed Systems.*

129. **[SM92]** A. D. Stoyenko, T. J. Marlowe, "Polynomial-Time Transformations and Schedulability Analysis of Parallel Real-Time Programs with Restricted Resource Contention," *J. Real-Time Systems*, Vol. 4, No. 4, pp. 307 - 329, November 1992.

130. **[SHH91]** A. D. Stoyenko, V. C. Hamacher and R. C. Holt, "Analyzing Hard-Real-Time Programs for Guaranteed Schedulability," *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, pp. 737 - 750, August 1991. Earlier version in the *IEEE RTSS'87.*

131. **[SBAM96]** A. D. Stoyenko, J. Bosch, M. Aksit and T. J. Marlowe, "Load Balanced Mapping of Distributed Objects to Minimize Network Communication," *J. Parallel and Distributed Processing,* 34(2), 117-137, May 1996.

132. **[SG92]** A. D. Stoyenko, L. Georgiadis, "On Optimal Lateness and Tardiness Scheduling in Real-Time Systems," *Computing,* Vol. 47, pp. 215 - 234, 1992.

133. **[SMYP97]** A. D. Stoyen, T. J. Marlowe, M. F. Younis, P. V. Petrov, "A Language Support Environment for Complex, Distributed Real-Time Applications," Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems, Milan, Italy, September 1997. Nominated for a Special IEEE Transactions of Software Engineering issue by Program Committee, Guest Editor and Area Editor of the IEEE Transactions of Software Engineering. Extended version to appear in the IEEE Transactions on Software Engineering, December 1998.

134. **[S92]** A. D. Stoyenko, "Evolution and State-of-the-Art of Real-Time Languages," J. Systems and Software, Vol. 18, pp. 61 - 84, April 1992.

135. **[S88]** K. Sycara. "Resolving Goal Conflicts Via Negotiation," In Proc. of the Seventh Nat. Conf. on AI (AAAI-88), St. Paul, MN., 1988

136. **[S89]** K. Sycara. "Multi-agent Compromise Via Negotiation," In M. Huhns and L. Gasser, editors, Distributed Artificial Intelligence, Volume II. Pittman Publishing Ltd and Morgan Kaufmann, 1989.

137. **[S91c]** K. Sycara. "Problem Restructuring in Negotiation," Management Science, 37(10):1248--1268, 1991.

138. **[S90]** K. Sycara. "Cooperative Negotiation in Concurrent Engineering Design," In D. Sriram, editor, Cooperative Product Development. Springer - Verlag, 1990

139. **[S90b]** K. Sycara. "Negotiation Planning: An AI Approach," European Journal of Operational Research, 46:216--234, 1990.

140. **[S90c]** K. Sycara. "Persuasive Argumentation in Negotiation," Theory and Decisions, 28:203--242, 1990.

141. **[SL91]** K. Sycara and M. Lewis. "Forming Shared Mental Models," Proceedings of the Thirteenth Annual Conference on Intelligent Agents, Monterey, CA, 1991.

142. **[SL98]** K. Sycara and M. Lewis. "Calibrating Trust to Integrate Intelligent Agents into Human Teams," the 31st Hawaii Systems Conference (HICSS-98), Hawaii, January 5-9, 1998.

143. **[S97]** K. Sycara. "Using option pricing to value commitment flexibility in multi-agent systems," Technical Report, School of Computer Science, Carnegie Mellon University, 1997.

144. **[SDPWZ96]** K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. "Distributed Intelligent Agents," IEEE Expert, 11(6), Dec. 1996.

145. **[SDZ98]** K. Sycara, K. Decker and D. Zeng. "Intelligent Agents in Portfolio Management". In Agent Technology: Foundations, Applications, and Markets], N.Jennings and M. Woolridge (eds). Chapter 14, Springer 1998, pp. 267-283.

146. **[SRSF91]** K.Sycara, S. Roth, N. Sadeh, and M. Fox. "Distributed Constrained Heuristic Search," IEEE Transactions on System, Man and Cybernetics, 21(6):1446--1461, 1991.

147. **[SZ96]** K. Sycara and D. Zeng. "Coordination of Multiple Intelligent Software Agents," International Journal of Intelligent and Cooperative Information Systems, 5(2 & 3):181--211, 1996.

148. **[TBK96]** D. Tayler, J.P. Black, T. Kunz. "Poet, Shoshin, MANDAS," *working papers (Web-available)*, University of Waterloo, 1996.

149. **[TS98]** J. Thomas and K. Sycara. "Stability and Heterogeneity in Multi-agent Systems," In ICMAS-98, Paris, France, July 1998.

150. **[TBYS96]** J. J. P. Tsai and Y. Bi and S. J. H. Yang and R. A. W. Smith "Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis," New York, John Wiley & Sons, Inc., 1996.

151. **[TMW91]** K. Trivedi, J. Muppala and S. Woolet, "Real-Time-Systems Performance in the Presence of Failures," *IEEE Computer*, Vol. 24, No. 5, pp. 37-47, May 1991.

152. **[TV91]** K. Trivedi, M. Veeraraghavan, "An Improved Algorithm for Symbolic Reliability Analysis," *IEEE Transactions on Reliability*, Vol. 40, No. 3, pp. 347-358, August 1991.

153. **[TMWH92]** K. Trivedi, J. Muppala, S. Woolet and Boudewijn R. Haverkort, "Composite Performance and Dependability Analysis," *Performance Evaluation*, Vol. 14, Nos. 3-4, pp. 197-216, February 1992.

154. **[TC93]** K. Trivedi, Gianfranco Ciardo, "A Decomposition Approach for Stochastic Reward Net Models," *Performance Evaluation*, Vol. 18, No. 1, pp. 37-59, July 1993.

155. **[TH93]** K. Trivedi, Boudewijn R. Haverkort, "Specification and Generation of Markov Reward Models," *Discrete-Event Dynamic Systems: Theory and Applications*, Vol. 3, pp. 219-247. 1993

156. **[TC93]** K. Trivedi, P. F. Chimento, Jr., "The Completion Time of Programs on Processors Subject to Failure and Repair," *IEEE Transactions on Computers*, Vol. 42, No. 10, pp. 1184-1194, October 1993.

157. **[TIC93]** K. Trivedi, Oliver C. Ibe and Hoon Choi, "Performance Evaluation of Client-Server Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 11, pp. 1217-1229, November 1993.

158. **[TTM93]** K. Trivedi, L. Tomek and J. Muppala, "Modeling Correlation in Software Recovery Blocks," *IEEE Transactions on Software Engineering (special issue on Software Reliability)*, Vol. 19, No.11, pp. 1071-1086, November 1993.

159. **[TMMK94]** K. Trivedi, Jogesh Muppala, Varsha Mainkar, Vidyadhar Kulkarni, "Numerical Computation of Response Time Distributions Using Stochastic Reward Nets," *Annals of Operations Research*, Vol 48, pp. 155-184, 1994.

160. **[TGMT94]** K. Trivedi, Robert Geist, Varsha Mainkar, Lorrie Tomek,"Reliability Modeling of Life-Critical Real-Time Systems," *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 108-121, January 1994.

161. **[TM94]** K. Trivedi, Manish Malhotra, "Power-Hierarchy of Dependability Model Types," *IEEE Transactions on Reliability*, Vol. 43, No. 2, pp. 493-502, September 1994.

162. **[TL94]** K. Trivedi, D. Logothetis, "Dependability Evaluation of the Double Counter-Rotating Ring with Concentrator Attachments," *IEEE Transactions on Networks*, Vol. 2, No. 5, pp. 520-532, October 1994.

163. **[TCK94]** K. Trivedi, Hoon Choi, Vidyadhar Kulkarni, "Markov Regenerative Stochastic Petri Nets," *Performance Evaluation*, Vol. 20, nos. 1-3, pp. 337-357, 1994.

164. **[TSP95]** K. Trivedi, Robin Sahner, Antonio Puliafito, "Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package," *Kluwer Academic Publishers* (418 pages), 1995.

165. **[T98]** G. Tecuci. *Building Intelligent Agents*. New York: Academic Press. 1998.

166. **[UAFSB98]** U.S. Air Force Scientific Advisory Board. *Report on Information Management to Support the Warrior*. U.S. Air Force SAB-TR-98-02. 1998.

167. **[V99]** K. Vicente. *Cognitive Work Analysis : Toward Safe, Productive, and Healthey Computer-Based Work*. New York: Lawarence Erlbaum. 1999.

168. **[WKSR94]** Waheed, A., B. Kronmuller, Roomi Sinha, and D. T. Rover, "A Toolkit for Advanced Performance Analysis," Proc. of Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '94) Tools Fair, Durham, North Carolina, pp. 376–380. 1994

169. **[WR95]** Waheed, A., and Diane T. Rover, "A Structured Approach to Instrumentation System Development and Evaluation," to appear in *Proceedings of Supercomputing '95*, San Diego, California, December 1995.

170. **[W95]** M.Wellman. "Market-oriented Programming: Some Early Lessons," In S. Clearwater, editor, Market-Based Control: A Paradigm for Distributed Resource Allocation. 1995.

171. **[WDS96]** M. Williamson, K. Decker, and K. Sycara. "Unified Information and Control Flow," In Proceedings of the AAAI-96 Workshop on Theories of Action, Planning and Control: Bridging the Gap, Portland, Oregon, August 1996. AAAI

172. **[WSG91]** Y.-H. Wei, A. D. Stoyenko, and G. S. Goldszmidt, "The Design of a Stub Generator for Heterogeneous RPC Systems," *J. Parallel and Distributed Computing*, Vol. 11, pp. 188 - 197, March 1991.

173. **[W99]** G. Weiss. "Learning to Coordinate Actions in Multi-agent Systems," In M. Huhns, & M.P. Singh (Eds.), *Readings in Agents.* San Francisco, CA: Morgan Kaufmann. 1999.

174. **[WG99]** G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* Cambridge, MA: MIT Press. 1999.

175. **[WW98]** W.E. Walsh and M.P. Wellman. "A Market Protocol for Distributed Task Allocation," In Proc. of ICMAS-98, Paris, France, 1998.

176. **[YMS94]** M. F. Younis, T. J. Marlowe, A. D. Stoyenko, "Compiler Transformations for Speculative Execution in a Real-Time System," *IEEE 1994 Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1994.

177. **[YTMS95]** M. F. Younis, G. Tsai, T. J. Marlowe, A. D. Stoyenko, "Formal Verification for Speculative Execution in Real-Time Systems," *First IEEE International Conference on Engineering of Complex Computer Systems,* 1995.

178. **[ZS96]** D. Zeng and K. Sycara. "Bayesian Learning in Negotiation," International Journal of Human-Computer Studies, 48, in press 1998.

179. **[ZS97]** D. Zeng and K. Sycara. "Benefits of Learning in Negotiation," Proc. AAAI-97, Providence, Rhode Island, U.S.A., July 27-31 1997.

180. **[ZS99]** D. Zeng and K. Sycara. "Dynamic Supply Chain Structuring for Electronic Commerce Among Agents," In M. Klusch, editor, Intelligent information agents. Springer, 1999.