(12)

Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control

Analysis and Simulation

Randall Steeb, Stephanie Cammarata,
Sanjai Narain, Jeff Rothenberg,
William Giarla

RAND

The research described in this report was sponsored by the Defense Advanced Research Projects Agency under RAND's National Defense Research Institute, a Federally Funded Research and Development Center supported by the Office of the Secretary of Defense, Contract No. MDA903-85-C-0030.

Library of Congress Cataloging in Publication Data

Cooperative intelligence for remotely piloted vehicle
fleet control.

"Prepared for the Defense Advanced Research
Projects Agency."
"October 1986."
Bibliography: p.
"R-3408-ARPA."
1. Artificial intelligence—Military applications.
2. Military surveillance—Mathematical models.
3. Vehicles, Remotely piloted—Mathematical models.
I. Steeb, Randall, 1946- . II. Rand Corporation.
III. United States. Defense Advanced Research
Projects Agency.
UG479.C66 1986 623.74 86-17767
ISBN 0-8330-0749-1

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

R-3408-ARPA

Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control

Analysis and Simulation

Randall Steeb, Stephanie Cammarata,
Sanjai Narain, Jeff Rothenberg,
William Giarla

October 1986

Prepared for the
Defense Advanced Research
Projects Agency

RAND

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER R-3408-ARPA — 2. GOVT ACCESSION NO. A174 506	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control: Analysis and Simulation	5. TYPE OF REPORT & PERIOD COVERED Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. Steeb, S. J. Cammarata, S. Narain, J. Rothenberg, W. D. Giarla	8. CONTRACT OR GRANT NUMBER(s) MDA903-85-C-0030
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Rand Corporation 1700 Main Street Santa Monica, CA 90406-2138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency DoD, 1400 Wilson Boulevard Arlington, VA 22209	12. REPORT DATE October 1986
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No Restrictions

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence
Problem Solving
Cooperation
Simulation
Remotely Piloted Vehicles

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see reverse side

DD FORM 1 JAN 73 1473

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A

— 2

by either humans or machines,

Groups of agents (human or machine) can solve shared tasks effectively by applying cooperative intelligence. Cooperative behavior, is necessary for solving problems that, because of time or other physical constraints, cannot be solved by one agent acting alone. Complex, spatially distributed military systems, such as tactical air operations, Naval task force control, and command and control, communications, and intelligence networks frequently rely on cooperative problem solving. This report develops aspects of cooperative intelligence in the context of a specific application, coordination of coordinating groups of ~~remotely piloted vehicles in a~~ RPV's in a surveillance mission. The findings suggest that (1) a combination of object-oriented simulation and logic programming appears to provide an effective framework for exploring and implementing distributed problem-solving systems, and (2) choice of task negotiation procedure, message passing protocol, planning algorithm, and uncertainty representation technique depends strongly on situational conditions such as time stress, communication costs, and number of planning options. (See also R-3158-AF, N-1854-1-AF, and N-2139-ARPA.)

B

PREFACE

This report summarizes the results of an analytic and experimental investigation of techniques for distributed problem solving, conducted for the Information Processing Techniques Office, Defense Advanced Research Projects Agency (DARPA), under Rand's National Defense Research Institute (NDRI). The NDRI is a Federally Funded Research and Development Center sponsored by the Office of the Secretary of Defense. The work focuses on the development of organizational structures for cooperative planning in complex, spatially distributed systems, using remotely piloted vehicle control as an illustration. The report should be of interest to researchers in distributed problem solving, concurrent processing, and large-scale simulation. Related research is reported in the following Rand publications:

P. Klahr, J. W. Ellis, Jr., W. D. Giarla, S. Narain, E. M. Cesar, Jr., and S. Turner, *TWIRL: Tactical Warfare in the ROSS Language*, R-3158-AF, September 1984.

D. McArthur, P. Klahr, and S. Narain, *The ROSS Language Manual*, N-1854-1-AF, September 1985.

R. Steeb, D. McArthur, S. J. Cammarata, S. Narain, and W. D. Giarla, *Distributed Problem Solving for Air Fleet Control: Framework and Implementations*, N-2139-ARPA, April 1984.

SUMMARY

Groups of agents (human or machine) can solve shared tasks effectively by applying what is known as *cooperative intelligence.* Cooperative behavior is necessary for solving problems that, because of time or other physical constraints, cannot be solved by one agent acting alone. Complex, spatially distributed military systems, such as C^3I (command, control, communications, and intelligence) networks, tactical air operations, and Naval task force control frequently rely on cooperative problem solving. In this report, we develop aspects of cooperative intelligence in the context of a specific application, coordination of groups of remotely piloted vehicles (RPVs) in a surveillance mission. This application is particularly suited to the use of distributed artificial intelligence (DAI) techniques, since it is characterized by limited communications, frequent coordinated behavior, and repeated dynamic replanning. We use the RPV task simulation to develop DAI techniques for solving key problems in task distribution, communication networking, situation assessment, and cooperative planning. We also examine support environments needed for the development of cooperative intelligent systems for graphics interfacing, activity scheduling, and multiprocessor interaction.

Our work with RPV fleet control simulations has led to the following initial conclusions:

- A combination of object-oriented simulation and logic programming appears to provide an effective framework for exploring and implementing distributed problem solving systems.
- Choice of task negotiation procedure, message passing protocol, planning algorithm, and uncertainty representation technique depends strongly on situational conditions such as time stress, communication costs, and number of planning options.

Preliminary recommendations for these issues are presented in the context of RPV missions.

CONTENTS

I. INTRODUCTION

Cooperative intelligent systems are coordinated groups of agents involved in distributed problem solving activities. Distributed problem solving has generated great interest recently with the advent of small, powerful microprocessors and reliable communication networks and the increasing awareness that many military problems are highly distrib uted in nature.

Many of the problems addressed by cooperative intelligent systems are not amenable to conventional techniques for "distributed processing," in which tasks are decomposed into independent computations and offloaded to multiple processors to run in parallel. This methodology is seldom applicable to complex military problems with spatially distributed tasks involving agents with inconsistent views and dynamically changing interdependencies. In these situations, interprocessor interactions can be as important as the independent calculations.

Distributed problem solving is substantially more complex than single-agent problem solving, although it uses many of the same tools. Multiple agents may have differing skills or expertise to contribute to the overall task. Thus they may have to decompose the overall problem into subtasks and negotiate them. The individual agents typically have limited knowledge, and each may maintain its own incomplete and possibly inconsistent database. Multiple agents must also coordinate task execution. For example, before an attack can be mounted to destroy an airfield, the surveillance and defense-suppression roles must be successfully completed.

Aspects of distributed problem solving have been studied in many diverse areas, including robotics (Davis, 1981; Konolige, 1981; Lozano-Perez, 1983), ground-based intelligence gathering (Lesser, 1981), tactical planning (Steeb and Gillogly, 1983), command/control simulation (Klahr et al., 1984), satellite communications systems (Kahn et al., 1978), and concurrent language design (Jefferson and Sowizral, 1982). Many of these applications exhibit limited communications, multiple inconsistent databases, requirements for coordinated behavior, and frequent dynamic assessment and replanning. The most difficult problems primarily concern task assignment, communication management, data fusion, and cooperative planning.

Over the past several years, we have abstracted the characteristics of several military and civilian air fleet control problems, postulated and examined many possible architectures for cooperation (Steeb et al., 1981), and implemented a series of increasingly complex demonstration systems (Steeb et al., 1984). The initial work dealt with simulated air traffic control (ATC) tasks and resulted in the specification of some heterarchical and hierarchical architectures for intervehicle coordination. This early work also included the development of an experimental framework for implementing and comparing alternative structures (McArthur et al., 1982). Using this framework, we experimentally examined means for negotiating task responsibilities among agents and scheduling activities within individual agents (Cammarata et al., 1983).

We eventually shifted our emphasis to the more complex and demanding problem of military remotely piloted vehicle (RPV) fleet control. Here the aircraft must perform a common mission, operate in a highly uncertain and hostile environment, and function under extreme communication constraints. All of these conditions make distributed problem solving both more difficult and more essential than it is in civilian ATC.

There are several additional, pragmatic reasons for concentrating on RPV coordination, most of them having to do with the potential gains distributed artificial intelligence (DAI) may provide over current control techniques. First, conventional radio control of RPVs from a command center may be ineffective under battlefield conditions because of heavy channel demands, frequent time delays, and high operator skill requirements. Local onboard control by intelligent processors lessens reliance on the electronic umbilical. Second, the tasks performed by RPV groups tend to be naturally distributed, both spatially and functionally. Surveillance, for example, requires aircraft to cover different regions, and different aircraft may perform entirely different functions—e.g., sensing, jamming, or relaying. The RPV fleet control task is also suited to DAI because of the existence of a body of codable expert knowledge, much of which is in the form of heuristics (closed-form analytic models have been defined for only a few aspects of RPV coordination). Finally, RPV fleet control is of current military importance. Several airborne military platforms already have sufficient payload capacity and electronic sophistication to support automated planning and control. Breakthroughs would have immediate effects on the development of new weapons systems, communication networks, sensor systems, and operational tactics. In Appendix A, we describe four key RPV missions—surveillance, defense suppression, communications relaying, and electronic warfare—and show how distributed problem solving techniques can improve system performance in each of these missions.

Our research does not focus on specifying more efficient and capable RPV fleets, however. We are primarily interested in developing new distributed problem solving techniques. Our first objective was to develop a general testbed for implementing distributed, cooperative systems. ROSS, the Rand object-oriented simulation system (described in Sec. III), provides an effective core for such a testbed. Second, we wanted to produce a series of RPV demonstration systems, providing an environment for developing new DAI techniques and illustrating and communicating our findings. Our demonstration system simulates RPV surveillance operations. Third, we wished to specify new DAI techniques covering such functions as task assignment, communication networking, situation assessment, cooperative planning, and the human interface. These techniques are applicable to a wide range of areas beyond RPV coordination. Section II provides a general description of the DAI techniques, and Sec. III examines their implementation in the surveillance RPV simulation. Finally, the techniques are applied to the creation of preliminary RPV operational guidelines, described in Sec. IV. Appendix A describes the RPV task domain and analyzes the potential usefulness of distributed problem solving techniques to key missions. Appendixes B and C describe ancillary implementation issues concerning multiprocessor communications and logic programming techniques.

II. DISTRIBUTED ARTIFICIAL INTELLIGENCE: RESEARCH ISSUES

Over the past few years, we have developed a series of increasingly complex civilian ATC and military RPV simulations that allow us to examine how individual agents can interact to accomplish common goals.

Our early ATC work was concerned primarily with methods of scheduling the various processes in each agent—communication, situation assessment, planning, and control—and comparing protocols for task negotiation among agents. We found that knowledge-based forms of scheduling, in which processes compete for attention and can interrupt each other, are often essential in a distributed task (McArthur et al., 1982). We also found that effective task allocation among agents depends on such disparate factors as agent knowledge, action flexibility, and communication load (Steeb et al., 1984).

There are many ways of decomposing a task and assigning the subtasks to processors. Each agent may control a different geographic region (as in current ground-based ATC), a different function (such as phases of flight—launch, enroute, targeting, and recovery—in military RPVs), a different possible path in a plan, or a different physical aircraft (Steeb et al., 1981). We have thus far concentrated on the last architecture, which we term object-centered. Here, each aircraft has onboard capabilities for all planning and control activities, providing the most general distribution form and the most immediate application possibilities.

In the ATC applications and later in our RPV work, we assumed that each aircraft was able to sense in a limited area, with some overlap between adjacent members of a group. We assumed that communications between group members were range-limited and frequently costly in some form (e.g., available bandwidth, resulting delays, enemy detection and jamming). Because of the limited communications and sensing, each aircraft developed its own uncertain and somewhat inconsistent model of the environment and of the plans of the other aircraft. This independent mode of agent operation is very different from the shared-memory approach frequently assumed in distributed processing systems. Shared-memory multiprocessor systems ensure database consistency either by directly accessing the same memory (using a common blackboard for making changes) or by having high-bandwidth communications so that all changes can be broadcast to the local databases. Most physically distributed military systems do not have these luxuries, however. In some fashion, they must balance database consistency with communication load.

We have implemented a series of surveillance RPV task simulations, beginning with a very rudimentary three-RPV task with no defenses or communication degradation, and evolving through versions with five RPVs, jamming, aircraft attrition, special roles, and group reorganization. These demonstration systems are discussed in detail in Sec. III.

We used the sequence of simulation systems to develop and explore distributed problem solving techniques for:

1. Distribution of tasks among processors.
2. Communication networking.
3. Situation assessment and data representation.
4. Cooperative planning and control.
5. User interface.

Each of these areas is discussed below.

DISTRIBUTION OF TASKS AMONG PROCESSORS

Assignment of tasks to processors is a very pervasive problem in distributed problem solving. The group must determine how to partition the overall task into relatively independent subproblems, negotiate (with a minimum of messages) the association of tasks and processors, and combine the resulting partial solutions into a coherent whole. Aspects of this divide-and-conquer process can be found in such disparate activities as automatic programming, operating systems scheduling, communications networking, corporate decisionmaking, and team sports. Over the years, several formal paradigms have emerged for distributed processing systems. The Contract Net and ESP approaches, for example, assigned processors to tasks on the basis of bids by the competing processors (Davis and Smith, 1981; Singh and Genesereth, 1984). This typically required a central decisionmaker to receive and compare the bids.

In our early simulation systems, the task negotiation process involved only selection of the RPV group leader. Each group member sent its estimates of its own suitability to be leader (termed the role factor) to the current leader. The current leader then compared the role factors and announced the new leader. This protocol worked adequately because there was no attrition (the current leader was never damaged or destroyed), and messages were not jammed or otherwise degraded. We found that the advantage of such a centralized organization over more anarchic forms increased as the number of aircraft increased (Steeb et al., 1984).

The picture changed when we went to more complex and realistic situations. The introduction of specialized roles, communication degradation (due to jamming or noise), and group losses and reorganization all forced modifications in the task negotiation protocols. For example, the negotiation for leadership might use very different criteria from those for data fusion, communication relaying, or other roles. Leadership requires a central position in the group, open communication links, and knowledge of defenses and group plans, whereas data fusion might require an outside position and specific defense knowledge. If jamming is present or losses occur, the communication protocols for negotiation must be modified. The aircraft require more robust approaches than centralized comparison, e.g., a preestablished line of succession or direct exchange and negotiation of individual role factors.

A change in the form and protocols for distribution of tasks may dramatically change the behavior and effectiveness of the group. In our early implementations, all situation assessment and planning functions were performed by the leader. The followers sensed data, sent it to the leader, responded to commands, and, when necessary, negotiated new leadership. If some of the leader's functions were subsumed by the followers, a speedup would occur through parallelism, although it would be offset to some degree by communication delays. Greater distribution of functions should allow some preprocessing and filtering of information prior to planning, and the group should be less vulnerable to loss of the leader.

COMMUNICATION NETWORKING

Communications between aircraft are often limited in RPV missions because of bandwidth problems, jamming, noise, delays, and the possibility of enemy interception. The RPVs must decide what messages to send, how to format them, and when to send them; they must also decide how to service incoming messages. In some instances, the RPVs must base these communication decisions on a model of the recipient's needs, balancing the costs of communication with the value of the data. Also, a multihop transmission path may be used, with the aircraft relaying messages, sending back acknowledgments, and indicating when a link is open, jammed, or dangerous.

Communications management problems become paramount when several RPVs must interact, and when jamming and noise are present. Here the RPVs frequently do not have a direct transmission path to the other aircraft. They must route messages through intermediate links, finding the most direct, most secure, and least loaded path, much like a packet radio system (Kahn et al., 1978). We considered three methods of routing messages between RPVs: communication tables, route setup packets, and spreading activation. Communication tables are lists maintained by each RPV that indicate which links are open, out-of-range, noisy, jammed, dangerous, or simply unknown. The lists may be updated when messages are received, acknowledgments are not returned, jamming is sensed, or other inferences can be made about the links. The second method uses route setup packets, special short-length messages sent along candidate routes prior to a normal communication. When acknowledged, they provide a means for comparing the immediate performance of each possible path, but they tend to burden the channel. The third method, spreading activation, involves sending copies of a multihop message along all possible pathways. This increases redundancy, virtually assures receipt, and requires no table management overhead, but it can severely tax channel capacity. We chose the communication table method in our early work, because of the high costs of communication in surveillance RPV operations.

The routing problem is basically a constraint-satisfaction task, so it seemed an appropriate task for which to compare LISP and Prolog implementations. The algorithms for each of these implementations attempt to find the lowest-cost (shortest, least dangerous, least noisy) route for each message. Prolog algorithms show advantages in simplicity of code and ease of memory management. Appendix A provides a more detailed comparison of the two languages, along with some examples of code.

Noise and errors in received messages lead to special problems. We assume that a receiving RPV can identify errorful messages through a parity check or other diagnostic procedure. The RPV can then request a retransmission. Also, RPVs can request retransmission of "lost" messages by noting omissions in message numbering sequences. For the most part, noise and errors produce only a variable delay in the transmissions.

Communications traffic can become particularly unreliable in highly stressed situations, such as movement through dense defenses with rapid attrition of aircraft. To alleviate this problem, we are examining more distributed and robust forms of transmission, such as stepwise relay, where each RPV determines the best link along which to send the next transmission. The sender thus designates only the recipient and the first hop. If the message is blocked, the process backtracks a step, and the previous sender updates its communication table and sends the message to a new RPV. This technique takes time but should result in more robust message transmission in degraded situations.

We have also considered several alternative protocols for the actual exchange of information. For example, the sender may infer a need by the recipient and transmit the data, or the receiver may directly request data from another RPV. These are termed volunteer and demand forms of communication (Lesser and Corkhill, 1983). In the volunteer form, a Prolog or LISP program may be used to infer what the other aircraft know, augmented by rules for deciding if the knowledge justifies the cost of transmission. The demand form can use a similar protocol to determine the best sources of information, or may simply involve interrogating all other aircraft when data are needed. The volunteer form is more efficient when information needs are known (Steeb et al., 1984). Examples of some inference rules are given in Sec. III.

SITUATION ASSESSMENT AND DATA REPRESENTATION

The RPVs must maintain estimates of defense locations and types and of the other RPVs' plans, status, and positions. Uncertainties in these estimates may arise due to sensing fallibility, communication noise, *a priori* data inaccuracy, or inappropriate rule application. Assertions about the defenses and the other RPVs can be represented probabilistically using LISP attributes, or Prolog Horn clauses. Which representation to use depends on how the data will be manipulated.

Much of the information about the defenses must be estimated using heuristics. Consequently, we used LISP attributes in our implementations for representation, and a MYCIN-like model of uncertainty for representing beliefs (Steeb et al., 1984). Each sensing contact by the RPVs adds to the measure of belief (MB) about a particular defense at a particular location. If the location is subsequently overflown with no contact, the RPV will add a measure of disbelief (MD). The confidence factor (CF) for a particular defense is then MB − MD; it runs from −1 to +1, although negative CFs are suppressed in all behaviors (i.e., no actions are triggered from negative information). When data are communicated, the confidence measures are modified according to the characteristics of the channel. The RPVs use similar CFs to keep track of their likelihood of having been detected by specific defenses.

Much of the information about the status of other RPVs, on the other hand, can be obtained through inferential reasoning and can therefore be represented and manipulated by Prolog statements. The status, communication links, and knowledge of another aircraft can be represented by Horn clauses, and Prolog routines will then determine if a piece of information will be useful to that aircraft. The Prolog routines will also infer many items of information, so that data requests should be reduced.

The defenses, meanwhile, perform the same functions of situation assessment and planning that the RPVs do. The ground control intercept (GCI) and surface-to-air missile (SAM) sites detect RPVs, send data to the command centers (CCs), and respond to commands to jam communications or intercept RPVs. For simplicity, we have not implemented a cooperative planning capability equivalent to that of the RPVs in these objects. Instead, we used a simple set of production rules to guide their behavior.

The MYCIN approach to uncertainty propagation works adequately for a few RPVs and a few objects, but it results in problems when many contacts are made or many expectations are violated. More rigorous approaches, such as the Dempster-Shafer calculus (Garvey et al., 1981; Gordon and Shortliffe, 1985), will be necessary for subsequent investigations.

COOPERATIVE PLANNING AND CONTROL

Planning, in the context of RPV fleet control, is the generation, evaluation, and selection of group maneuver options. The maneuvers may be necessary to avoid defenses or to collect surveillance information. This planning function is closely tied to some of the communication management and situation assessment actions, such as deciding whether to send or withhold information, or whether to infer or request data about the defenses. In fact, the planning options may even include waiting to receive more information through sensing, communication, or inference.

In our early demonstration systems, planning was of a straightforward condition-action form. The RPV leader created a model of the defenses and the possibility of enemy detection and used this model to match triggering conditions to maneuver actions. (Some illustrative rules are described in Sec. III.) In this paradigm, the RPVs do not project the group's trajectories, ascertain performance and danger, or select the best plan. They simply match the situational conditions (defensive positions, probability of detection, time since last contact) to one or more communication or maneuvering responses.

Our early demonstration systems also had a somewhat restricted action set. The RPVs could change coverage pattern (racetrack or figure-8), formation type (wave, vee, or stream), spacing (wide or close), and leadership in response to threats or opportunities. But as we considered more options and made the situational conditions more complex, we were led to develop a multistep planning process with pruning of options and simulation-based evaluation. Such an approach deemphasizes detailed situational specification and gives added importance to accurate simulation and evaluation (Hayes-Roth, 1985). A possible sequence of steps is the following:

1. *Problem recognition.* The leader projects the current flight plan of the group and notes a problem: high danger or low probability of additional surveillance information.
2. *Time-constraint check.* The leader determines the time until an action must be taken. This planning interval must be greater than the estimated time to generate alternatives, project them, evaluate choices, select one, command others to take actions, and execute the plan. If insufficient time is available, the leader terminates the planning process and selects the best immediate situation match.
3. *Option generation.* If the planning interval is sufficient, the leader finds all the maneuver options applicable to the situation. It then culls these to a viable set by checking constraints, such as distance from boundaries, formation requirements, and previously used actions.
4. *Option projection.* The leader projects each remaining candidate option and records expected outcomes: detections, jamming, losses, and coverage. The depth of projection may depend on the time available and the database confidence. The leader may distribute the projection task by requesting each follower to project its own trajectory and communicate the result.
5. *Option evaluation.* The leader weighs the projected losses and coverage and chooses the best option.
6. *Option execution.* The leader commands the followers, giving maneuver commands and constraints to monitor. The leader also responds to messages from the followers when constraints are violated, modifying the plan accordingly.

This sequence is an elaboration of a planning process used in our earlier ATC work (Cammarata et al., 1983). The leader first determines that the problem is important enough to go through deep planning, and that there is sufficient time to complete the planning process. Then, he and the other RPVs generate possible options, pruning them by checking available constraints. The RPVs then project the options forward to evaluate their effectiveness. The leader compares the options and selects one for execution. Of course, these activities do not all have to be performed by the leader; they may be distributed across the group.

USER INTERFACE

A particularly pervasive problem throughout our research has been the user interface—the "window" on the workings of the many separate agents. At a minimum, the user must be apprised of the activities being performed by each agent, the current situation assessments, the communications being passed, and the planned trajectories and actions. He must also be able to focus his attention on specific areas, controlling such functions as pan, zoom, time-stepping, and level of detail.

Our initial implementations used a C-based graphics system which was designed primarily for information display rather than user input. The simulator, at the end of an update, would transmit data needed to produce the next display frame to the graphics program. The graphics program would display the frame and allow such operations as zoom and pan, but would not allow the user to point at objects or request information. Some queries could be made though an auxiliary textual screen, but these required direct input to the simulation.

Ideally, a simulation should display its simulated world with appropriate detail and continuity to allow visual comprehension of the behavior of the underlying model. It should allow interactive control over what is animated and should permit stepping through animated events. It should allow a user to interact graphically with the displayed image to pan and zoom, change the positions of objects (for example, to define an initial configuration or scenario), etc. Ultimately it should allow direct graphical interaction to affect the simulation itself (e.g., drawing trajectories with a "mouse").

It is useful to allow the creation of "graphic artifact" objects which do not correspond to real-world objects but which enhance comprehensibility. Such objects often depend on (simulated) real-world objects or on other graphic artifact objects. For example, a communication between two simulated entities is shown by a line between them, the endpoints of which must be determined by the positions of the communicating objects at the time the communication is displayed. Similarly, the radar-sensing envelope of a group of sensing objects is displayed as a curve which merges the individual envelopes of those objects at their positions at the time of display. In such displays, positions must be represented symbolically so that they can be evaluated at the time of display. This kind of graphic dependence should be handled by the system without requiring the programmer to worry about the order in which things are evaluated.

To allow this kind of graphic interface to be built, we provided a flexible graphics-in-LISP (GIL) facility that would use the graphics kernel standard (GKS) or Core graphics standards packages available on Sun workstations. This was done to provide an appropriate medium-level graphics capability in LISP with a minimum of effort. A lower-level, device-oriented graphics facility was rejected because it would require inordinate effort to provide sufficient power for our application, and its nonportability would limit future evolution. A higher-level, simulation-oriented facility was rejected on the basis that we could not know in advance how it might evolve and therefore could not define it appropriately. We chose a medium-level facility, because it would allow the use of LISP to explore future approaches to graphic simulation, and the GKS and Core standards would provide a reasonable medium-level model that has been subjected to wide critical review. Although GKS and Core are not identical, they provide similar functionality (the GIL facility is designed to use either one to provide equivalent functionality). Both standards provide device-independence, resulting in a high degree of portability. The implementation itself is described in greater detail in Sec. III.

III. SIMULATION METHODOLOGY

We explored many of the issues raised earlier by implementing a series of simulation systems. It readily became apparent that producing a testbed for simulation of multiple autonomous agents is a major research effort in itself. Our implementations evolved through single- and multiple-processor systems and through many different languages. In this section, we first describe our object-oriented approach and then discu~s some of our experiences and recommendations.

AN OBJECT-ORIENTED APPROACH

The simulation routines, the planning and problem solving procedures, and many of the graphics facilities for our RPV simulations were written in ROSS, an object-oriented simulation language developed at Rand (McArthur, 1984). Programs written in object-oriented languages consist of a set of objects that interact with each other via the transmission of messages. Each object (e.g., RPV, radar, SAM site) has a set of attributes describing itself, and a set of message templates and associated behaviors. A behavior is invoked when a message is received that matches a template; the typical behavior would be message transmissions to other actors. The object-oriented style aids the understanding of distributed problem solving systems, because objects control their own activities through individual behaviors and maintain their own models via their local databases. For example, data about sensed defenses are represented in the vehicle's database, and reactions to the defenses result from behaviors triggered by the receipt of messages. Our implementations employ this object-oriented structure in three distinct areas of processing: behaviors for simulating the scenario, behaviors for cooperative planning and control by the RPVs, and graphics behaviors for user display and interaction.

The simulation behaviors define aspects of the scenario and capabilities of the objects. These behaviors include defining RPV trajectories, specifying time increments, calling randomization programs, sensing other objects, communicating messages, and calculating outcomes. We defined special objects for some of these functions, such as communication, scheduling, and performance monitoring, since these processes were considered operational requirements rather than problem solving activities.

The second type of processing consists of behaviors for cooperative planning and control by the RPVs. These behaviors include rules for reasoning about role assignments, making decisions about generating and routing messages, synthesizing maneuvers, and monitoring execution of the plans. ROSS is well-suited to expressing such behaviors, because most of them are already in the form of responses to messages. For example, the English and ROSS versions of a behavior for avoiding defenses are shown below:

English:

If the group is in a stream formation, and the leader estimates the probability of detection is greater than .6, then change to another coverage pattern.

ROSS:

```
(If (and
    (eq (~your formation ) 'stream)
    (~you are leader)
    (greaterp (~your probability-of-detection) .6))
 then
    (~you change coverage-pattern))
```

The third type of processing concerns the graphics environment. In our early systems, this environment was programmed in a combination ROSS- and C-based subsystem. Communication with the simulation objects was performed in ROSS, while device-dependent operations were implemented in C. The system was able to display task conditions, individual aircraft behaviors, and rule firings. As described earlier, we programmed some of the graphics routines in LISP, so that the operator could make queries and changes directly.

The three types of processing combine to produce a relatively general distributed problem solving testbed. We were able to implement a series of simulations with different message passing protocols, data representation forms, scheduling and control structures, and analysis routines. These are described in some detail in Sec. IV.

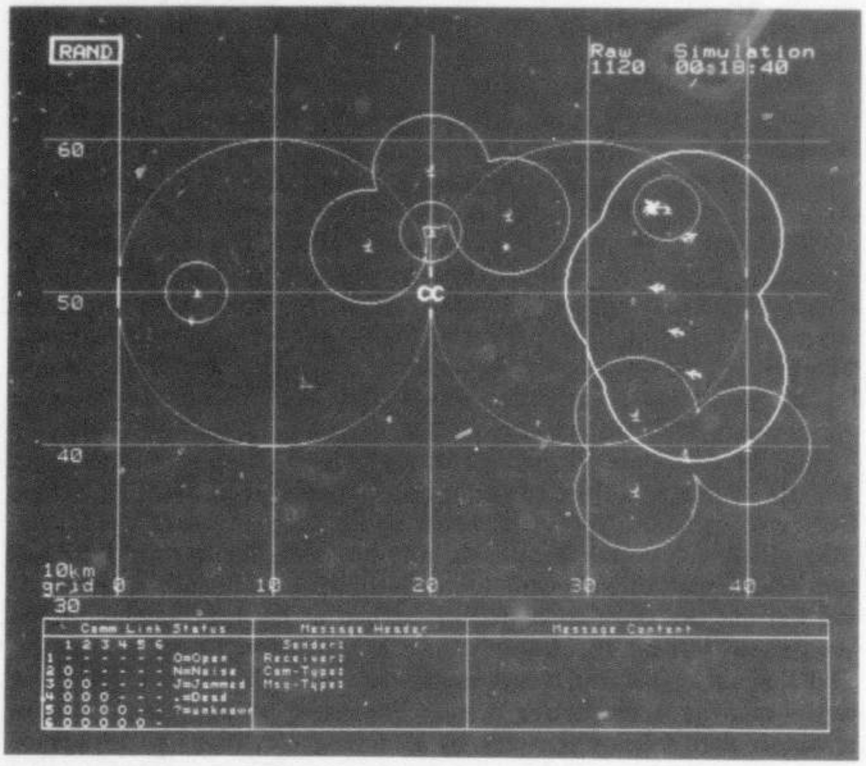
The typical form of the task simulation is shown in Fig. 1. Five RPVs gather intelligence about ground defenses in a hostile area. The RPVs then change their coverage pattern (the large circular and figure-8 patterns in the figure), their formation geometry, and several other aspects of flight in response to enemy threats or surveillance opportunities. The defenses themselves can interact to detect, jam, and fire on the RPVs; GCI radars, for example, can detect an RPV and send a message to a nearby command center, which can then call for jamming by the GCI site or launch a SAM. In Fig. 1, GCI radars are shown as small radar dishes surrounded by their sensing radii; SAM sites have smaller sensing radii; and CCs and airfields have none. Figure 1 shows an explosion as the lead RPV is hit by a SAM, and Figs. 2, 3, and 4 show the reorganization of the group as its members interact to avoid the defenses, negotiate leadership, and assign data fusion responsibilities.

In our research, the simulation task passed through several phases, each exercising a different aspect of cooperative situation assessment and planning. Throughout the remainder of this report, we refer frequently to these phases, using them as contexts for discussing distributed problem solving methods. We began by implementing a three-aircraft demonstration of patterned flight over a benign environment. This initial phase was useful for defining communication protocols and leader negotiation procedures. These investigations are described in our previous report (Steeb et al., 1984).

The second phase introduced active ground defenses and sensing. The RPVs could, within a certain range, sense GCI sites, SAM sites, CCs, and airfields. Certain defenses could likewise detect the RPVs. This phase was useful for establishing protocols for passing information about the defenses and for orchestrating avoidance maneuvers. We did not use the normal ROSS method of message passing in the simulation; instead we created a new ROSS object for each message. The channel object could then manipulate the message in a variety of ways—relaying it, corrupting it, etc. The leader would respond to data messages and defense sensings by matching conditions to avoidance maneuvers, which would be carried out by sending commands to the followers.

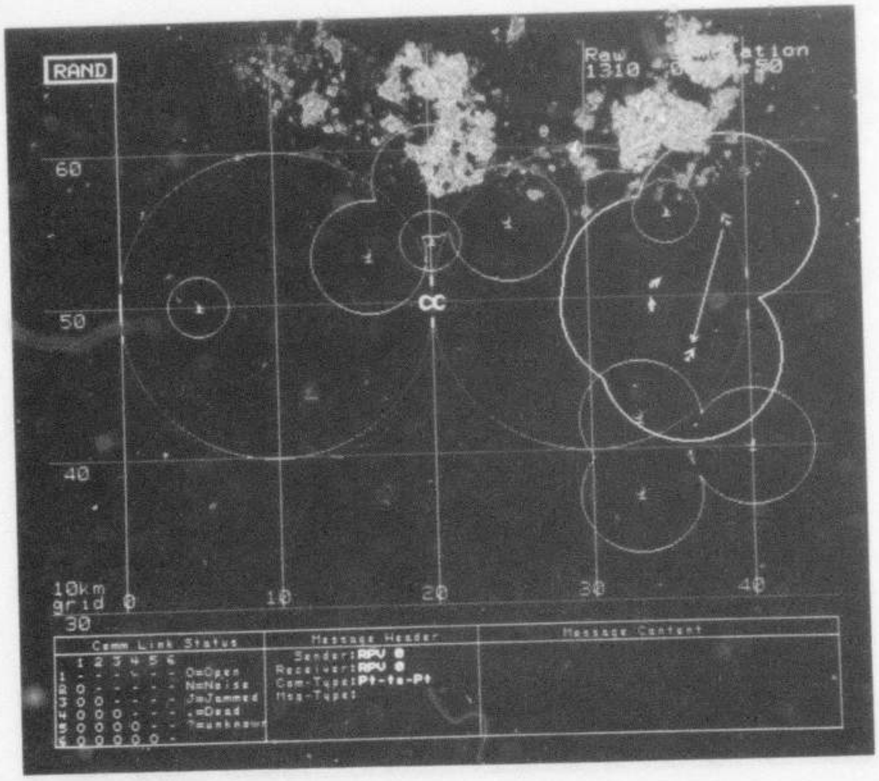
In the third phase, we expanded the group to five RPVs and added uncertainty to the processes of sensing by the RPVs and detection by the GCI and SAM sites. The larger number of RPVs made the interactions more complex and the likelihood of relaying messages greater. The addition of uncertainty led to implementation of a MYCIN-like system for maintaining and updating hypotheses in the database. Measures of belief, disbelief, and confidence were maintained for assertions about defense location and type, and about whether the RPVs had been detected by the defenses. The subsequent addition of jamming by the GCI and SAM sites led to the introduction of communication tables, with each aircraft sending messages among the group by relaying them over several aircraft.

The current implementation (the fourth phase) adds special roles and inference capabilities to the RPVs, coordinated attack functions by the defenses, and reorganization of the RPVs with losses. Most of these functions are implemented in the form of ROSS rules. For



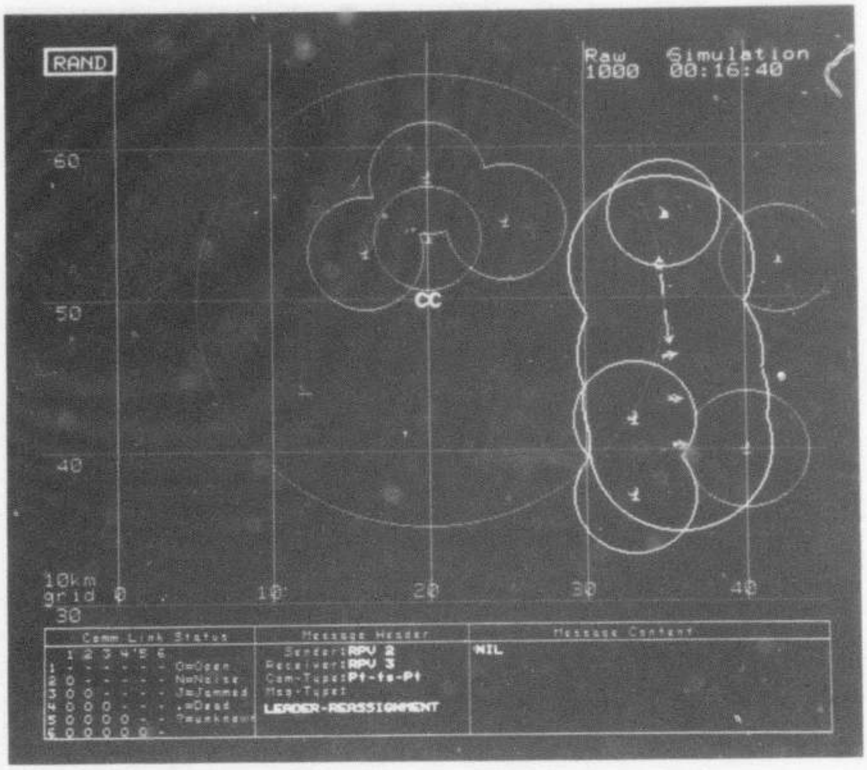
```
  Comm Link Status         Message Header    Message Content
   1 2 3 4 5 6              Sender:
1  - - - - - -  O=Open     Receiver:
2  O - - - - -  N=Noise    Com-Type:
3  O O - - - -  J=Jammed   Msg-Type:
4  O O O - - -  .=Dead
5  O O O O - -  ?=unknown
6  O O O O O -
```

Fig. 1—Graphics display for RPV task simulation, showing RPV explosion



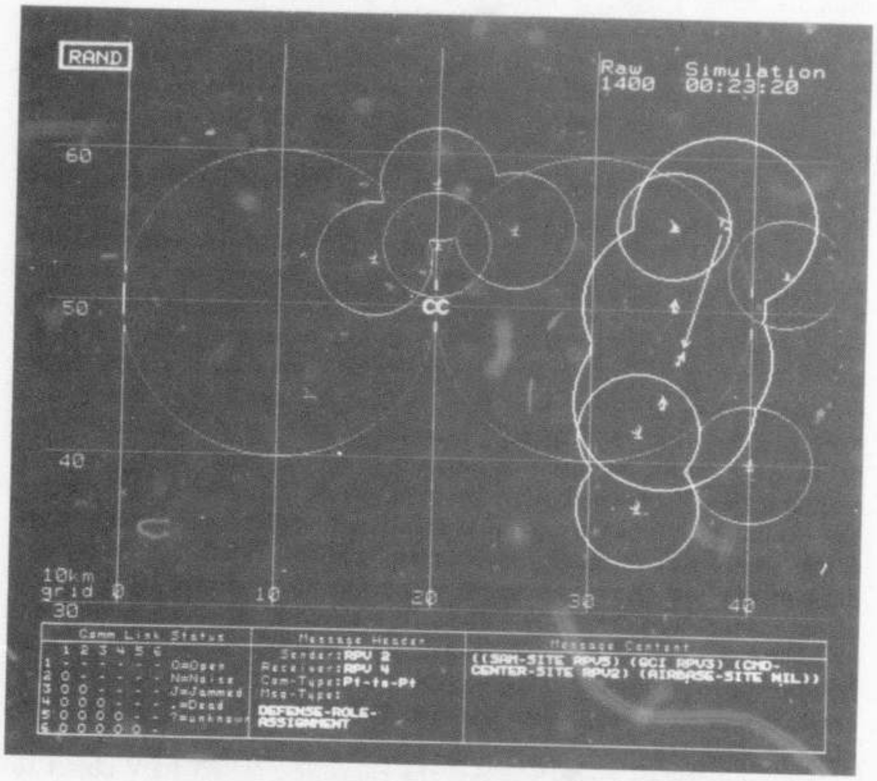
```
Comm Link Status         Message Header         Message Content
  1 2 3 4 5 6             Sender:RPV 0
1 - - - - - -  O=Open    Receiver:RPV 0
2 O - - - - -  N=Noise   Com-Type:Pt-to-Pt
3 O O - - - -  J=Jammed  Msg-Type:
4 O O O - - -  .=Dead
5 O O O O - -  ?=unknown
6 O O O O O -
```

Fig. 2—Graphics display showing survivors regrouping following loss



```
Comm Link Status              Message Header            Message Content
 1 2 3 4 5 6                  Sender:RPV 2              NIL
1 - - - - - -  O=Open         Receiver:RPV 3
2 O - - - - -  N=Noise        Com-Type:Pt-to-Pt
3 O O - - - -  J=Jammed       Msg-Type:
4 O O O - - -  .=Dead         LEADER-REASSIGNMENT
5 O O O O - -  ?=unknown
6 O O O O O -
```

Fig. 3—Graphics display showing leader reassignment



```
        Comm Link Status
   1 2 3 4 5 6
1  - - - - - -   O=Open
2  O - - - - -   N=Noise
3  O O - - - -   J=Jammed
4  O O O - - -   .=Dead
5  O O O O - -   ?=unknown
6  O O O O O -
```

Message Header
Sender: RPV 2
Receiver: RPV 4
Com-Type: Pt-to-Pt
Msg-Type:
DEFENSE-ROLE-ASSIGNMENT

Message Content
((SAM-SITE RPV5) (GCI RPV3) (CMD-CENTER-SITE RPV2) (AIRBASE-SITE NIL))

Fig. 4—Graphics display showing negotiation over data-fusion roles

example, when triggering conditions are present, the RPVs negotiate responsibilities for leadership and for data fusion about specific types of defenses. The defenses, in turn, are given specific behaviors. The GCI sites send contact information to the command center, which then commands the nearest SAM sites to fire on the RPVs sighted. If losses are sustained, the RPVs use negotiation and line-of-succession rules to reorganize. We also augmented the graphics interface to show more of the RPV interactions.

Over time, the sophistication of our simulation knowledge and the scenario configuration both increased. As discussed above, the scenario was expanded from three to five vehicles, the defenses in the system became active firing agents instead of passive sensing objects, and additional forms of communication networking were added, among other enhancements. We had developed a complex system of interacting agents, and within each agent, a complex repertoire of problem solving skills. As a result, we found it difficult to analyze the effectiveness of adding a new RPV capability. On an absolute scale, we could analyze the overall performance of the simulation to determine whether outcomes were being improved or deterred by a new capability. However, most of the new knowledge being added was accompanied by a variety of parameters and threshold values. We found it increasingly difficult to isolate the effects of a new capability for selecting parameters and fine-tuning our choice of threshold values. Furthermore, in many cases, it was advantageous to exercise a new strategy in isolation from other capabilities.

For example, we recently added the capability for an RPV object to perform an individual maneuver. Previously, all actions by RPVs were instructed by the leader or by a set of "group negotiated" plans. The new knowledge for independent RPV maneuvers would enable an RPV to decide to temporarily diverge from the fleet. Incorporating this knowledge led to questions about when to perform independent maneuvers, what the maneuvers should look like, who should be notified of the independent actions, and when the RPV should rejoin the fleet. We designed and implemented behaviors to effect individual RPV maneuvers, which included many variable parameters, such as time of divergence and trajectory of the maneuver. Our initial experimentation resulted in some unexpected behavior. Isolated scenarios that we presumed would perform badly were actually offset by other knowledge such as fleet avoidance, defense responsibilities, and fleet reconfiguration. We observed that other knowledge embodied by an RPV sometimes compensated for unintelligent decisions about individual maneuvers. Through similar repeated experiences, we realized that we needed better methods for isolating and identifying effects of new knowledge.

We developed a reduced RPV scenario which provided only the very basic navigational, sensing, and communication skills. This reduced configuration provided an environment which could be viewed as a kernel debugging tool for experimenting and testing object behaviors and capabilities. We first reduced the number of RPVs from five to two to limit the complexity and combinatorics of RPV negotiation. Whereas the original scenario contained three SAM sites, six GCI sites, and a CC, the defenses in the reduced scenario were limited to two SAM sites. This reduction helped to minimize reactive behavior by the RPVs. In the complete surveillance scenario, RPVs are rarely in a quiescent state; they are constantly reacting to defense proximities in a prioritized fashion. By enabling periods without defense interference, we could better observe inter-RPV exchanges, which had previously been masked by high-priority RPV/defense exchanges. A third modification involved trajectory simplification. In the large-scale scenario, RPVs navigated on either a figure-8 or a circular track plan. In the small-scale scenario, the trajectory was reduced to a simple linear track extending diagonally across the surveillance airspace. Normally, figure-8 and circular trajectories are advantageous because of the iterative coverage they offer. However, we found that repeated sightings of defenses resulting from track plan iterations increased an RPV's probability of detection. This increased detection in turn resulted in higher RPV attrition, which is undesirable in a testing environment. The fourth simplification we implemented minimized the communication traffic by allowing only point-to-point transmission. This restriction, coupled with a reduction in the number of RPVs, substantially eased the message traffic and allowed RPVs to respond in a timely fashion to messages. Without these limitations, an RPV's input message queue was rarely flushed. Furthermore, the input messages were prioritized on the basis of urgency, which resulted in some of them being outdated by the time they were received.

With our reduced simulation system, we were able to test new RPV capabilities much more effectively. We noticed that the kernel system streamlined our development, debugging, and testing process in general, and promoted good software engineering practices. We found that it was much easier to expand the reduced scenario to add more sophistication than it was to iteratively reduce the full-scale simulation system.

We were aware, however, of the dangers of using this methodology. First, we did not want to reduce the scenario so drastically that the problems we were addressing would become trivial and unrealistic. We were also concerned that we might ignore the effects of integrated behaviors and skills by focusing strictly on single capabilities. Finally, we wanted to ensure that a scaled-up version would be valid. Our solution was to retain and incrementally upgrade the large-scale scenario with the full complement of RPV facilities. In this way, we utilized the limited system to focus and pinpoint results for local debugging and sensitivity analysis, but relied on the full-scale sophisticated system for testing overall performance and outcomes.

In all these simulations, we made a number of simplifying assumptions and abstractions. Our primary goal was to produce a useful testbed for developing and exploring DAI techniques, not to reproduce a hypothetical, high-fidelity RPV mission. For example, we represented time and space continuously in some calculations, but most behaviors involved stepping time forward in discrete (but variable) steps. We simulated behaviors such as sensing, jamming, and communication probabilistically, using a randomization program in the scheduler object. Finally, we constrained some of the complexity of the problem by making the airspace two-dimensional. Altitude was not represented explicitly, and we ignored the problem of inter-RPV collision during maneuvers (we assumed that each RPV was at a different assigned altitude). We felt that the resulting simulation was both rich enough and sufficiently constrained to allow us to examine techniques for distributed problem solving.

LANGUAGE CONSIDERATIONS

Our original ATC simulations were programmed in ROSIE[1] and then in LISP. As we moved to the more complex domain of military RPV control, we found that we needed a more powerful and efficient simulation language. We chose ROSS because its message-passing formalism is well-suited to modeling interactions among several objects. ROSS also has advantages over other simulation languages, such as Simscript (Kiviat et al., 1968) and Simula (Dahl and Nygaard, 1966), because of its object-oriented nature, its inheritance features, its pattern-matching flexibility, and its English-like form for representing rules. ROSS also enabled us to use a substantial amount of code that had previously been developed for air-penetration and ground-warfare scenarios (Klahr et al., 1982; Klahr et al., 1984).

The demands on the simulation language increased dramatically as the research program evolved. Initially, our emphasis was on role negotiation and activity prioritization, functions that entailed relatively

[1]ROSIE is the trademark and service mark of The Rand Corporation for its Rule-Oriented System for Implementing Expertise.

simple messaging, situation assessment, and planning operations. We later expanded these functions to include complex data fusion, communication relaying, reorganization, and planning activities. As a result, the number and size of rules increased rapidly, with a consequent slowing of code generation and simulation speed.

Our development process in the ROSS programming environment was slowed by poor execution speed, lack of flexibility during object and behavior specification, and the need for manual record keeping. The execution-speed problem was due to the multilayered nature of ROSS. ROSS is written in LISP, and therefore ROSS code can be executed in either an interpreted or compiled mode. Also, it is easy (and encouraged for some tasks) to integrate LISP procedures and functions with ROSS behaviors. The existence of the two languages, although compatible in the interpreted mode, contributed to some semantic problems when running compiled code. We eliminated this incompatibility by rewriting portions of ROSS and aspects of the compiling environment. The second obstacle resulted from a stringent convention in ROSS requiring objects and behaviors to be unnaturally distributed across files. This convention restricted the modularity of the code, making debugging and testing difficult. We alleviated this problem by adding some file-management routines to ROSS. The third deficiency concerned bookkeeping and file-management operations which had to be performed manually. By automating these operations, which was fairly easy, since they are quite mechanical, we increased the speed of compiling and executing ROSS code.

We also added some capabilities to ROSS by coding some functions in Prolog (written in Franzlisp), a logic programming language that is better suited than LISP for certain constraint-satisfaction tasks, such as routing of messages during jamming, information-value calculations, RPV slot reassignments, and special inferences. For these functions, Prolog provided advantages over LISP in efficiency of expression and memory management. Appendix A presents some empirical comparisons between the two languages.

We may eventually have to further augment the current message-oriented structure of ROSS to include special database demons, plan scripts, and other procedures. Database demons would be used to test for complex combinations of conditions. This may be more efficient than the current ROSS procedure of spreading separate message templates across many objects. Similarly, plan scripts (time-tagged sequences of conditional actions organized much like frames) may provide a better means of representing timing and sequencing than the "send an action reminder to yourself" procedure of ROSS (Carbonell, 1978; Steeb and Gillogly, 1983).

GRAPHICS INTERFACE

Our original graphics interface used the ROSS graphics package (RGP), which had been modified repeatedly to adapt to changes in host hardware, display hardware, and UNIX operating system versions. The graphics hardware interface for the RGP was written in C and called from LISP (ROSS); this allowed ROSS objects to display themselves by calling functions that accepted their graphic attributes as parameters. This approach was necessitated by the lack of graphics support in Franzlisp; it had the fortuitous advantage of hiding the details of the display hardware from the simulation, while allowing the C code to take advantage of the hardware and provide a device-independent interface to the simulation. The attendant disadvantage was that the simulation had only limited and indirect control over the display. In addition, changes in display hardware often required drastic changes to the C code.

Since LISP provides only minimal support for calling "foreign" programs (such as the graphics object-code routines), we decided to build a fairly general interface facility to allow importing such foreign code into LISP. This facility should eventually also provide transparent multiprocessor communication across a net to allow experimenting with distributed architectures and offloading graphics processing onto dedicated machines. Toward these goals, we have implemented the first phase of what we call "hoses" (which resemble UNIX "pipes" but are more flexible). The GIL facility uses hoses to access the SunCore package from Franzlisp. Hoses are described in detail in Appendix B.

GIL is oriented around the concept of an image composed of a number of graphic "segments" (analogous to those implemented by the GKS and Core standards). A segment can contain an arbitrarily complex picture composed of many individual items; it encapsulates this picture, allowing it to be displayed, hidden, transformed (i.e., scaled, rotated, moved), etc. An application can therefore change some aspects of the displayed image simply by changing attributes of segments (and the transformations applied to them) rather than by redisplaying the entire image. The application can also change an overall transformation that applies to the entire image (e.g., for panning or zooming). In short, GIL maintains the state of the display and allows efficient incremental changes to be made to it. Although it is possible for an application to redisplay the entire image whenever it changes anything (as occurred with the RGP), this undermines the efficiency of the graphics standards (and also prevents them from utilizing the power of newer graphics machines that employ dedicated graphics processors to perform segment transformations in hardware).

GIL also interfaces with input devices such as the ubiquitous mouse. When the user selects an object on the screen, GIL tells the application which segment (and which item within the segment) was selected. The application maintains a mapping between segments and objects to determine which object the user selected.

Display-Update Alternatives and the Graphics Delta Approach

The basic purpose of simulation graphics is to display the behavior of a model. (The term "model" is used here to refer to the underlying simulation independent of any graphic display.) We have identified three approaches to keeping a display up to date with respect to the model. The first is to let the model run without performing any explicit graphics, and to use a "graphics engine" to update the display by continually redisplaying the entire state of the model; this is analogous to maintaining a bit-mapped (or character-mapped) display with a "display processor" (essentially the approach implemented by the RGP). The second alternative is an "incremental graphics" approach, in which the model updates the display explicitly whenever anything changes. The third alternative is a compromise between these extremes in which the model performs no explicit graphics, but the graphics engine figures out what must be changed for each new graphics frame and generates incremental updates as needed. We call this the "graphics delta" approach.

The advantage of the display processor approach is that the model need not be concerned with producing graphics output; it simply runs and keeps its state current. The display processor keeps the simulated situation displayed by essentially redisplaying its entire state with each frame. The display processor must access graphic attributes of the model only when they are in a consistent state, and it must generate a new graphics frame whenever a "graphically significant" event occurs in the model, which requires synchronization with the model. In addition, every new frame requires redisplaying the entire model state.

In the incremental graphics approach, the model generates graphics output as it runs, thereby relying on a maintained graphics state which it updates as needed. This results in greater efficiency, which in turn may improve the appearance of the display, since if a given graphic attribute is not affected between frames it will not cause the display to be updated (this reduces redundant updating, which in turn reduces both graphics processing and potential visual distraction). It may also result in better dynamics: When an event in the model produces graphics output, the model can use special graphics techniques (e.g., animation) to highlight the meaning of the event for the user (this is more difficult with the display processor approach, since the meaning of the event may be lost by the time the display processor produces its next graphic update). The disadvantage of this approach is that the model must perform all graphics output explicitly; it must be concerned with graphics throughout its code. The conceptual cleanliness of simply asking each displayable object to display itself with each frame is lost.

The graphics delta approach combines the advantages of the other two approaches. The model performs no explicit graphics output, but simply updates attributes, as in the display processor approach. However, all "graphic attributes" (those that affect the display) keep a history of their last-displayed value. A new frame is generated whenever certain "graphic triggering" events occur, such as the changing of selected graphic attributes, clock ticks, and explicit display commands necessitated by the model. These graphic triggering events invoke the graphics engine implicitly (it does not run independently as it does in the display processor approach). The graphics engine, rather than redisplaying the entire simulated situation, uses the history of the model's graphic attributes to form a "graphics delta" between the current graphic state of the model and the last displayed frame. This graphics delta is used to generate incremental updates to a maintained graphics state, thereby retaining the efficiency of the incremental approach. In many ways, the resulting protocol is similar to that necessary for efficient communication between two problem solving agents. The sending agent transmits only the information that is essential to the recipient.

The graphics delta approach retains the conceptual simplicity of the display processor approach, since the graphics engine can simply ask each displayable object to redisplay itself; however, this redisplay has the efficiency of the incremental approach as a result of the maintenance of graphic-attribute history. Synchronization is provided by the implicit invoking of the graphics engine by the occurrence of graphic-triggering events in the model; these events are also responsible for forcing the model into a consistent state before each graphic update. The only potential deficiency of this approach is that (unlike the incremental approach) it may not always allow taking full advantage of the meaning of a graphics event within the model; such meaning must be captured by the model at the time of the event by means of setting additional attributes.

Autonomous Attributes and Update on Demand

Attributes that change over time (such as position) present special problems in a simulation, since the simulator must always ensure that they are up to date before using them (e.g., displaying a new frame). We term such attributes "autonomous" because they conceptually vary by themselves (as time advances). Autonomous attributes are normally updated as a function of time. To maintain consistency for such attributes, we have implemented an "update on demand" strategy which forces an autonomous attribute to be updated whenever its value is consumed within the simulation. This mechanism allows graphics updates to be performed without additional synchronization (beyond the triggering discussed above). In fact, this approach ensures consistency of autonomous attributes throughout the simulation (not just during graphic updates), thereby eliminating considerable ad hoc code from the model, which would otherwise have to perform explicit updates wherever there is any chance that they may be necessary.

Intermediate Mapping Layer

The approach outlined above assumes that when an object is asked to display itself, it can generate the delta between its current graphics state and its previously displayed state in such a way that the display can be updated efficiently. This requires that the graphics engine maintain a state that can be incrementally changed. In order to maintain the necessary correspondence between the states of objects and their displayed representations, an intermediate layer is interposed between the application and GIL. This layer maintains what is in effect a symbolic display list for GIL. It also handles those cases where an incremental change in an object's graphics state can be effected only by redisplaying the object (i.e., when changing display attributes that can be changed only by deleting and recreating segments). The intermediate layer also provides mapping between the graphics engine's representation (e.g., Core/GKS segments) and the application's objects.

Graphic Dependence and Graphic Artifact Objects

To handle the display of communication links between objects, radar-sensing envelopes, and similar phenomena, we have introduced the notion of a "graphic artifact object" whose image depends on other objects. Graphic artifact objects provide a way of representing "presentation artifacts" that have no direct analogues in the real world. The images of graphic artifact objects must always be computed late, because they may depend on position information about other objects; this prevents "ghost" images, for example, showing a communication link between previous positions of moving objects.

IMPLEMENTATION CONFIGURATION

Cooperative interactions among multiple intelligent agents can be simulated with a single processor or with a multiprocessor network. For much of our initial work, we chose the simpler single-processor route. The surveillance RPV task was programmed in ROSS and Prolog on a VAX 11/750 (later a Sun 2/120), with C-based color graphics running through an AED 512 (later in LISP/GKS on a Sun 2/160). All simulation objects (RPVs, GCI and SAM sites, CCs, communication channels, etc.) were treated as separate entities with their own databases. The entities shared processing resources, and time was stepped forward in such a manner that asynchronous behaviors were accurately reproduced. In fact, we felt that true asynchronous multiprocessing would not contribute significantly to the fidelity of the simulation, since all communication delays, timing problems, etc., could be programmed into the single-processor simulation. A multiprocessor system would, however, provide advantages in enhanced graphics interfaces and speedup through parallelism.

We also explored the possibility of transferring this single-processor implementation to a multiprocessor environment with two Sun workstations on an ethernet. Here we would dedicate one Sun as a simulation processor and one as a graphics processor. The initial architecture would consist of a one-way communication channel, from the simulator to the graphics, to be upgraded later to two-way communication for user input from the graphics side. The system would achieve speedup by having the simulation one or more time steps ahead of the graphics. The simulation Sun will be processing for time t while the display is being generated for time (t − n). This type of scheduling assumes there will be little or no intervention (beyond screen parameter inputs) from the user. As the need and demand for user interaction increase, it will be necessary to synchronize the simulation and graphics, either by backing up the simulation one or more steps or advancing the graphic display one or more frames. The choice will depend on the size of the time step, the granularity of the simulation, and the type of user interaction. This form of distribution should be totally transparent to the user. The user should be able to initiate, view, back up, interrupt, or restart the simulation from the graphics processor, unaware of the tandem configuration.

An alternative distribution would be to couple each RPV to an individual processor. Each processor would have its own graphic display, based on the contents of the database of the resident RPV. This configuration would be very useful for isolating and observing the behavior of a particular RPV under different conditions and with different roles. The viewer could interrupt the simulation, modify some parameters or behaviors, and watch the effect of the changes. A distribution of this sort would require some major extensions to the ROSS language for distributing objects and passing messages across machines.

With any of these multiprocessor configurations, the connections between machines become important, particularly since each machine may be running multiple processes in different languages. The hose facility (mentioned above) allows the processors to interact efficiently.

IV. CONCLUSIONS AND FUTURE WORK

We noted at the beginning of this report that we have four long-term program objectives—development of a general testbed for implementing distributed systems, implementation of a series of RPV demonstration systems, specification of new DAI techniques, and application of the techniques in the form of operational guidelines. We have made progress, although not always in the anticipated direction, on all of these objectives.

Our testbed has moved from a simulation of three RPVs flying over a benign environment to one with five RPVs experiencing jamming, uncertainty, losses, and reorganization. The testbed itself is composed of a combination of ROSS simulation functions, Prolog inference routines, and C- and LISP-based graphics. In this development, we found that ROSS is very well suited for distributed problem solving research. Its inheritance feature, message-triggered action, and English-like rules make it effective for transparently modeling interactions among a group of agents. At the same time, we found that Prolog is more effective for certain problem solving tasks (several examples are given in Appendix C).

A particularly challenging aspect of the testbed was the graphics interface. As the task simulation became more complex, far more information had to be displayed to the user: object locations and types, communication links, situation assessments, processor activities, group organization, etc. To follow the cooperative behaviors, the user must control the characteristics of the displays by setting the level of detail, time steps, and viewpoints, and making specific queries of the problem solving agents. These needs led us to develop the GIL and hose methodologies.

The difficulty of producing a coherent, understandable, and interactive testbed for simulation of multiple problem solving agents ultimately led us to initiate a new study of knowledge-based simulation methodology. This new study will augment the object-oriented ROSS simulation methodology with reasoning, explanation, and interactive graphics capabilities. We will attempt to develop representations that allow the user to view the model at different levels of abstraction, run separate portions of the overall simulation, and view graphic explanations of system behavior.

Our second product, the RPV fleet control demonstration system, provided a specific environment for developing and testing new DAI techniques. As we progressed, the RPV simulation became more complex and realistic, since the RPV agents had to contend with uncertainty, jamming, and losses. The RPVs had to cooperate to build up a picture of the defenses while minimizing their losses and maintaining their organization. The task combined aspects of situation assessment and cooperative planning that are common to virtually all complex military systems. It also displayed the many forms of communication degradation—noise, delays, jamming, and range problems—that make distributed problem solving essential. The new forms of organization and behavior we have explored should have significant impacts on RPV design, tactics, and doctrine.

The third product, new DAI techniques, was our main concern. We examined techniques for distribution of tasks among processors, situation assessment, communications networking, cooperative planning, and implementing the human interface. We have discussed each of these separately above, but in fact they are strongly intertwined. We also examined different paradigms for knowledge representation and activity scheduling.

Our work in task negotiation showed the importance of defining effective "role factors" to use in comparing each agent's suitability for tasks. The work also showed the need for specifying efficient communication protocols for exchanging role factors. The factors we noted as important to leadership an data fusion included aircraft location, communication link status, and knowledge of defenses and plans. Role negotiation should occur only when major disruptions occur—e.g., the group suffers losses or the leader is isolated—and communications are sufficiently open for the rest of the group to interchange factors. If the group negotiates too often, the communications and processing loads could be overwhelming. We also found that if the leader is still operational, the most efficient protocol is to have the leader request factors from the others, compare them, and announce the roles. When the leader is not able to do this or more robustness is necessary, each of the remaining agents can broadcast its role factors, with the first one receiving a full set announcing the new roles. In emergency situations, when links are degraded or time is critical, the RPVs may simply use a preset line of succession to designate their successors. An alternative for future exploration is a form of chain letter, in which the group circulates one or more role-factor messages, each aircraft adding its own factor and passing it on until the list is complete. Another possibility is a more free-form negotiation, in which the participants exchange messages describing tradeoffs and constraints.

Situation assessment encompasses the problems of data representation and probability aggregation. We found that the rudimentary

MYCIN formulation for representing and propagating belief and disbelief in an assertion was sufficient for work with a few objects and a few data points. As the number of objects, sensings, and data correlations increased, however, the MYCIN formulation began to have problems, because of its disregard of independence assumptions. Subsequent research should utilize a more rigorous approach, such as the Dempster-Shafer or Bayesian paradigms, to produce more accurate estimates of belief, disbelief, and ignorance in each proposition.

A sideline aspect of situation assessment is information-value calculation. When an RPV makes a sensing, changes its plan, or receives new information, it may calculate the information's value to another RPV before sending it to that RPV. Such a balancing of information value and communication cost is often necessary in noisy or dangerous situations or when there is much overlap of knowledge. We explored the use of Prolog programs for making such evaluations, by inferring what the other RPVs know and need to know. This representation form was found to be more compact and efficient for this task than LISP.

The third DAI issue we examined was communication networking. In this task, messages frequently have to be relayed across the group, e.g., when interaircraft links are out of range, noisy, or jammed. We found that an effective means of routing messages was to maintain a communication table in Prolog and use logic programming algorithms to "prove" that a viable path exists. We recommend extending this type of procedure in highly uncertain situations by having the sending aircraft decide which is the best aircraft to send to in the relay sequence, and letting that aircraft decide on the next step. This may occasionally require some backtracking.

The communications themselves involve data transmissions, role-factor exchanges, task announcements, action commands, and acknowledgments. Several of these can be performed in either a demand or volunteer form. For example, an RPV responsible for intelligence on GCI sites could either periodically interrogate the other aircraft or wait for them to send updates when they sense such sites. The demand form is more efficient during periods of communication danger, but it results in data fusion delays. It appears that algorithms are needed for selecting the communication protocols according to the danger and time stress present.

Most of the initial actions in our work in cooperative planning were group maneuvers chosen by the leader in response to situational conditions. We later expanded this planning activity to include projection and evaluation of several candidate plans over multiple updates. This required ancillary development of heuristics for determining whether sufficient time exists for the planning, evaluation, command, and execution cycle. This process is much more efficient in distributed form, with all the aircraft participating in the projection and evaluation. The planning may be done by assigning different candidate options to each aircraft for projection, or by having each aircraft project and evaluate its own flight path in a given option. We also added individual actions to the repertoire of RPV responses to danger, such as splitting off from the group and rejoining, which considerably reduced response time to threats, at the expense of occasionally decoupling the group.

The final DAI problem we addressed was the development of an effective user interface. The RPVs in our scenario were designed to perform all operations (communications, planning, etc.) autonomously, without human input. However, the user did act in a supervisory mode during development and needed to follow the reasoning used by the system. The user also needed to be able to change properties and behaviors when problems occurred. We found that a two-way graphics interface was essential for this, with the user able to point at an icon and alter it, query its state, or change its knowledge. The framework for this capability was developed with the GIL and hose constructs. We also found that animation of each aircraft's plan, showing projected problems and progress toward a solution, is essential. In future work, we hope to expand the interface functions to include graphic explanation capabilities and adjustable levels of abstraction.

Scheduling the many behaviors involved in situation assessment, planning, and control requires a few rules of its own. In our previous ATC work, we developed a special knowledge-based scheduling routine in which activities (input-communication, planning, output-communication, control) would vie for attention, and when invoked would still be subject to interruption by more important activities. Thus an aircraft could be in the midst of planning but would suspend the planning process to service an input message describing another aircraft's plan. Using coroutines, the system would then either continue or restart the planning process with the new information. Our early RPV work did not require interruption of activities that had been initiated. Instead, we used a situation-specific set of activity priorities to decide on the next activity to process, and each activity was then run to completion. In our later work, which involved occasional deep planning and many dynamic assumptions, we found more need for activity suspension and resumption.

A second type of scheduling we employed involved the simulation of multiple objects on the same processor. In our single-processor system, we simulated all the objects on the same machine, emulating asynchronous processing of multiple independent objects through the use of a special scheduler object, which switches processing attention and moves time ahead in such a way as to emulate asynchrony. In future work, we plan to assign individual RPVs to separate processors. The behaviors and interactions will be the same as in the current simulation, but multiple graphics views will be provided, along with some additional speedup through parallelism.

A major problem in our work has been that of planning and unplanning. It is difficult in an event-based simulation to simulate continuity without seriously compromising code efficiency. Event-based simulations often require explicit unplanning of events when assumptions change, since events are planned in the future based on the current state. As time elapses, some events become invalid, and the programmer must specify necessary changes resulting from object interactions. This compromises code modularity and puts a large burden on the programmer. We are currently exploring methods for automating the unplanning process.

In most of our work, we kept the architecture constant (leader-based with demand communications, heterarchic with volunteer, etc.) and varied the situational conditions, such as defenses, jamming, and losses. This approach revealed some things about the applicability of the different architectures to different situations, but only touched on how to transition dynamically between architectures during a mission. Such transitions require rules for selecting protocols as conditions change, sending bursts of information necessary to support the new member responsibilities, and monitoring performance under the new conditions. An RPV mission might involve an enroute phase with a leader-based organization and open communications, an intelligence gathering phase with anarchic organization and volunteer communications, and an attack phase with leader-based structure and demand communications.

It is still too early to coalesce more than a few application guidelines from our studies. In our primary application area, RPV fleet control, the appropriateness of DAI techniques depends on the degree of automation onboard the aircraft. At one extreme, the current method of RPV control involves almost continuous control by a human operator over a wideband communication link. Here, very little of our work is applicable. At the other extreme, the RPVs might pursue their mission without any human intervention. If they do so in a totally preprogrammed fashion, much like current-generation cruise missiles, only techniques such as communication relay of sensed data may be appropriate. Our work is primarily applicable in the case of RPVs that are both autonomous and intelligent, responding to situational conditions as they arise, or of RPVs that are controlled in a supervisory fashion by human operators but still able to assess situations and respond to them locally. Some preliminary application guidelines for these situations are given below. Specifics of some of the RPV missions discussed are given in Appendix A.

The first set of recommendations concerns role negotiation and reorganization with losses. Here, tasks are matched to RPVs on the basis of such factors as RPV location, knowledge, communication links, damage, fuel, and payload. The tasks may include group leadership, data fusion responsibilities, or special operations such as decoy, relay, or damage assessment. Prior to a mission, a command center can specify roles for each RPV, along with lines of succession in the event of damage or loss. (Such lines of succession have long been used in fighter and bomber squadrons to regroup following losses.) Unfortunately, preestablished successions do not take into account changes in member conditions accruing during the mission. For example, the PRV that is next in line for command may become damaged, jammed, or removed from the group. Reorganizing according to dynamic conditions requires some form of interaction, such as comparison and assignment by a designated aircraft or by inter-RPV negotiation. The choice of method depends on the channel characteristics, time stress, and amount of group dislocation. Some exemplary rules are:

1. If there is high time stress or minimal changes in status among surviving RPVs, or highly degraded communications (noise, delay, danger of detection), use *succession.*
2. If communications are only moderately degraded and the leader (or leader-designate) is operational, use *assignment* (the RPVs send any status changes to the designated aircraft, which then reassigns roles).
3. If there is free communication, low time stress, and only local dislocation (between only two or three RPVs), use *negotiation* (the RPVs use a back-and-forth protocol of comparing appropriateness for roles).

Communication, used in reorganization and many other processes, is itself a very complex function, encompassing the problems of what, when, and how to send messages. The RPVs may send wide- or narrow-beam transmissions directly to each other, they may use satellite relay, or they may rely on high-altitude radio relay. The messages themselves include data, commands, acknowledgments, and requests. The techniques we examined include whether to demand or volunteer information, and how to route messages among RPVs. The demand form of communication seems best suited for collecting role factors and requesting expectation checks, since these are triggered at one node (often without awareness by the others) and require directed input. The volunteer form appears best for sending sensed data to those responsible for processing those data and for sending action commands to other aircraft. Both forms can benefit from information-value checks (to determine that the value of the message is greater than the projected communication costs).

Message routing, the second aspect of communication, depends on time stress, channel reliability, and link status knowledge:

1. If communication channels are heavily jammed or unreliable, correct routing is unknown, communication costs are low, and time stress is high, use *spreading activation* (the sender transmits the message to all those in range, and all receiving the message attempt to pass it on, until the intended recipient receives it).
2. If correct routing cannot be established (due to insufficient channel status knowledge) and time stress is low or communication costs are high, use *chain letter* (the sender chooses the best first hop and sends the message, the next RPV choses the next hop, and so on, with backtracking if a blockage occurs).
3. If correct routing can be established, use a *routing table* to specify all intermediate relays on the way to the intended recipient.

The choice of communication-routing protocol thus depends strongly on the mission phase. During high-altitude surveillance operations, the RPVs will frequently exchange intelligence data. Decision time stress will normally be low, and aircraft status will be fairly constant, but there will be some likelihood of detection and jamming. Communication tables should thus be maintained, allowing fully specified pathways. Low-altitude operations in dangerous areas, on the other hand, will usually involve high time stress, frequent blanking of channels due to terrain and jamming, and rapidly changing status. Here the spreading-activation and chain-letter approaches would be more appropriate.

Planning in the RPV domain includes determining the best avoidance maneuver to take, changing course to cover new areas, waiting to gain more information, and changing formation to obtain supporting data on new contacts. The aircraft may plan for themselves individually or they may plan for others. Planning may be very rigorous, using logic programming or analytic techniques to "prove" that a solution exists, or very heuristic, using a set of incomplete or even inconsistent rules to arrive at a solution. Some guidelines are:

1. If time stress is high, situation knowledge is limited, or decision importance is low, use *pattern matching methods* to decide on the next action. Examples include jinking (rapid avoidance) maneuvers when a missile is sighted, changing formation after an absence of contacts, or making small course adjustments to maintain position in the group. The rules may be organized into sets associated with each situation type. Situation knowledge must be represented in a form that can be easily matched by rule antecedents.
2. If time stress is low, few options are present, substantial situation knowledge is available, and deep planning is necessary, use *simulation-based planning.* Here the aircraft simulate their own and enemy actions over the next several updates, checking constraint violations and goal accomplishments. This technique might be used for invoking pattern and formation changes to avoid known danger regions and to assure surveillance coverage of enemy areas. The plans would normally be preloaded as detailed sequences of action.
3. If there is substantial situation knowledge and many options, and the planning primarily involves constraint satisfaction, use *logic programming.* This technique might be appropriate for generating trajectories through the defenses, for controlling sensors, and for managing communications. It allows plans to be generated step-by-step from primitive actions.

Finally, we noted some considerations about group organization. In most situations, the leader-based structure is the most efficient. The leader gets the overall picture and is able to optimize group actions, and fewer messages are transmitted than with a distributed system. In fact, most command and control systems during peacetime operations are organized in this type of centralized, hierarchical structure. As the situation degrades, though, less efficient but more robust distributed organizations become favored. Data fusion tasks soon become clustered and localized as the RPVs become more separated. Emergency reactions to threats are initiated by the individual aircraft, and communication relays are channeled through any pathway available. The planning process is parceled out, first by having each aircraft simulate its own path forward, checking for constraint violations, and then by having each aircraft search out group options. The thresholds for moving through these progressive stages of distribution are not yet established and will be one of the key goals of future work.

Our explorations of cooperative intelligent systems will be continued in several very different research directions. We plan to expand our applications work to include all four of the RPV missions described in Appendix A—surveillance, electronic warfare, communications relaying, and defense suppression—along with implementations of ground robotic systems. We are also in the process of expanding our testbed through our knowledge-based simulation study, adding capabilities for hybrid representation forms, graphics explanation, and multiple levels of abstraction. Finally, we are beginning to examine concurrent-processing issues, such as load balancing, task scheduling, context switching, and synchronization.

Appendix A

BACKGROUND: OPERATIONAL RPV APPLICATIONS

RPVs controlled by a direct radio link or following some preprogrammed trajectory have proven to be useful in a variety of support roles. In this appendix, we will examine the RPV coordination problem and argue that expansions of RPV missions to include cooperative, autonomous action by intelligent RPVs will greatly increase their effectiveness. We begin with an overview of U.S., NATO, and Israeli RPV operations, followed by a discussion of the full range of possible RPV missions. We then focus on four missions that appear to have particularly high potential for cooperative interaction: surveillance, defense suppression, communications relaying, and electronic warfare (EW). These missions should benefit from intervehicle interactions that achieve optimum search and surveillance patterns, nonoverlapping target assignment, effective network positioning for communication, and multiaircraft jamming and deception capabilities. Finally, we describe how these four missions provide the organizational basis for a testbed for development and demonstration of DAI techniques.

RPV CAPABILITIES

RPVs have been used by U.S., NATO, Israeli, and Soviet forces for several decades. We use the term RPV somewhat loosely, to include both radio-controlled vehicles and autonomous unmanned vehicle systems (UVSs). RPVs run the gamut from small 50-kt sensor platforms to Mach 2 target drones. Some of the more important RPVs for our discussion are:

- *U.S. Army Aquila*: A small (140-lb) RPV with 3-hr endurance and 118-kt maximum speed. Planned missions include surveillance, target acquisition, artillery adjustment, and laser designation for precision guided weapons. Special configurations provide spread spectrum communications, automatic link loss reaquisition, and adjustable linking (high-G avoidance maneuvers) (Gossett and Velligan, 1982).

- *USAF Pave Tiger*: A canister-based, low-cost RPV with 100-kt cruise and up to 8-hr loiter capability. Sensor, electronic countermeasure (ECM), and warhead payloads are planned. Control is by preprogrammed autopilot or by radio link (Jane's, 1983–84).
- *USAF BQM-34 multimission drone*: A high-cost, high-performance (700-kt) radio command drone. Reconnaissance, EW, and warhead versions have been used. Weight is between 2500 and 5000 lb, and range is up to 700 n mi (Jane's, 1983–84).
- *Israeli Mastiff*: A small RPV used for battlefield and battle group surveillance. It weighs 250 lb and has a flight endurance of 6 hr (Hyman, 1981).
- *Israeli Scout*: A larger propeller-driven RPV with a takeoff weight of over 300 lb and a maximum cruising speed of 95 kt (Hyman, 1981). It has been used for surveillance with a stabilized TV camera and for decoy operations by electronically emulating larger aircraft.
- *British Army Phoenix*: A small RPV fitted with thermal imaging (infrared (IR) zoom) for both day and night surveillance (Klass, 1984).

In general, these RPVs have low radar cross-section, long loiter times, and moderate onboard processing capabilities. Payload is limited, though, so that normally they can perform only one type of mission at a time. The main advantages of these RPVs compared to manned aircraft are their mobility, acquisition cost, payload efficiency, endurance, modularity, life cycle cost, and expendability (Lupo, 1984).

RPV APPLICATIONS

RPVs have been proposed for use in a wide range of difficult and dangerous missions, including intelligence collection, data relay, reconnaissance, search and rescue, atmospheric sampling, electronic warfare, dispensing or dispersal of chaff, air-to-air defense, interception, surveillance, and airlift (Sanders, 1981). Some additional proposed missions include antisubmarine warfare (ASW), defense of a surface fleet, and support of tactical airstrikes against tanks. We will first examine the full range of missions and then focus in on four missions that have great potential for cooperative intelligent behavior.

Probably the most frequently mentioned RPV mission is battlefield surveillance. Here RPVs take the place of Army Aerial Observers (AOs) in light aircraft and helicopters, and of Air Force Forward Air

Controllers (FACs) in aircraft (Ellis, 1978). The minimum configuration is some form of imaging sensor and data link, tying the RPV, say, to an Army Air-Ground Operations System. Sensor types used include TV, photo, IR, signal intelligence (SIGINT), electronic intelligence (ELINT), and moving-target indication (MTI). The surveillance information can provide assessments of strength, composition, and axes of advance of the enemy. The Israelis have used RPVs, for example, for artillery spotting, forward area control, and battlefield management (Klass, 1984). RPV surveillance could also extend the range of existing systems, such as AWACS and E-2C, by flying at the limits of the sensing area and relaying data.

A second important mission, defense suppression, can involve either an explosive payload or use of large RPVs with onboard missiles. Direct attacks have been made by RPVs against enemy radars (using home-on-jam or home-on-emission). The RPVs loiter some distance behind the FLOT (Forward Line of Own Troops) and wait for radars or communications jammers to come on. Alternatively, active radar or passive IR sensors on the RPVs may be used to locate and home in on large GCI dishes and associated power sources (Ellis, 1978). Some of the larger RPVs, such as the Teledyne BGM-34B, have been used to launch Hobos guided bombs, Shrike antiradar missiles, and Maverick TV-guided missiles (Hyman, 1981). Even the small, recoverable Aquila has been fitted with 2.75-in. Viper recoilless antitank rockets (Gossett and Velligan, 1982).

A mission between surveillance and attack is weapon guidance. Acquiring and designating targets for the laser-guided Copperhead projectile, the Hellfire, the Maverick, and the multiple-launch rocket system (MLRS) is one of the key functions planned for the Aquila. RPVs may also be used for fire adjustment for conventional artillery.

Communications relaying is a straightforward RPV mission. Highly secure, broadband, jam-resistant communications packages using the JTIDS, ICNS, and other formats have been proposed for RPV use. The Aquila, for example, has a sophisticated Harris Corporation data link with two steerable EHF antennas. Also, multiple-access spread-spectrum systems have been developed using lightweight modems and surface acoustic wave devices. These have the advantages of low output signals and good interference rejection. The RPVs may position themselves for optimum line-of-sight transmission to the command centers, forming a robust network. This may require some inter-RPV coordination, including tracking of each RPV's position. The RPVs may also tie in with geostationary communications relaying platforms, although these platforms may be vulnerable to jamming. In either the RPV-only or the RPV-satellite configuration, long periods of autonomous RPV operation may be necessary because of both unintentional radio frequency interference and deliberate electronic countermeasures.

The other side of communications is EW, where denial and deceptive jamming are primary missions. A threat library of electronic discriminating characteristics can be preprogrammed into the sensor processor. Once a target is detected, determined to be a target of interest, and located by frequency/bearing, it can be jammed by one or more RPVs. An RPV can also accurately deliver expendable jammers to the other side of the FLOT. For deception, the electronic signature of an RPV can be electronically enhanced to provide a decoy mimicking high-priority helicopters or fixed-wing aircraft.

Coordinated behavior appears to be essential to many of the above missions. RPVs may need to be in proper relative position for relaying messages, for efficient surveillance coverage, and for multistatic sensing (in which one RPV might be transmitting and another receiving radar energy). Coordination is also necessary to assure nonoverlapping assignment to targets, and for efficient damage assessment and retargeting. Unfortunately, only a few multi-RPV control systems have been tested, and none of these has involved local coordination by the RPVs themselves. Long, vulnerable links to a centralized airborne or ground control center have always been present. For example, the Air Force outfitted a C-130 with drone control avionics capable of handling up to eight BGM-34C drones simultaneously. During tests, the controllers encountered problems with time-sharing of tracking, telemetry, and command functions (Klass, 1975). No formation flight was attempted. A ground-based, multi-RPV system was later created by IBM at White Sands Missile Range. This system involved formation flight of full-size F-102 drones and BGM-34 subscale drones, and coordinated movement of M-47 tank targets (Gray et al., 1982). Using special distance-measuring tracking and downlink telemetry, a central control facility was able to maintain formation, perform avoidance maneuvers, and follow complex trajectories. All decisionmaking was performed by the ground control center.

FOUR EXEMPLARY MISSIONS FOR SIMULATION AND TESTING

Four RPV missions appear to be good candidates for development and application of DAI techniques: surveillance, EW, communications relaying, and defense suppression. We shall describe some possible scenarios, tactics, and measures of effectiveness for these applications.

For simplicity, we concentrate on the use of mini-RPVs (under 300 lb and usually propeller-driven) in a NATO/Warsaw Pact conflict. These smaller RPVs have long endurance and, because of their low cost and projected use in large numbers, many opportunities for cooperative behavior.

Surveillance

Battlefield surveillance is used to obtain real-time assessments of enemy strength, composition, axes of advance, logistics, and tactics. As a possible scenario, assume a rapid Soviet armored attack along multiple axes, with deep thrusts into West Germany and continued pressure by commitment of second-echelon forces. Defense against such an attack requires timely and accurate surveillance of enemy operations, particularly of mobile Soviet air defense units (radar-controlled guns, radar-guided missiles, and IR-homing missiles).

A coordinated group of RPVs would require a ground or airborne supervisory station located close to the FLOT (to minimize range), and sufficient numbers of RPVs with imaging sensors to assure identification and classification of air defense units in a swath up to about 70 mi inside the FLOT. The sensor suite on a single RPV could be either a passive ELINT module and IR detector, or an active radar system. The RPVs would also need processing capabilities for data fusion, communications management, avoidance-maneuver initiation, and reorganization (following losses). The RPV fleet might also have to alter its course to concentrate on certain areas after achieving initial contacts.

RPVs operating cooperatively in a multistatic mode (one or more RPVs transmitting and one or more RPVs receiving the reflected signals) should be particularly effective in a jamming environment. The location of the silent receiver RPV would be unknown to the threat jammer, and narrow RPV scanning beams would allow the receiving RPV to detect those targets not screened by jammers in one or more of the possible multistatic radiation scattering directions (Henderson, 1982). Also, wide separation between RPVs would permit accurate direction finding and passive ranging on the jammers.

The deep penetration provided by RPV fleets should result in many tactical advantages. Strikes on rear logistics units should provide great leverage. With more accurate data on enemy positions, manned strike aircraft should be able to employ less vulnerable flight profiles and delivery tactics than would otherwise be required to acquire, identify, and attack battlefield targets. When comparing cooperative RPV performance and that of conventional tactics in this type of mission, the primary figure of merit should be the accuracy and timeliness of the situation assessment achieved.

Electronic Warfare

Electronic countermeasures in the European theater will involve both denial (jamming enemy communications and radar sensing) and deception (presenting false targets). The closely related attack functions of home-on-jam and home-on-emission will not be considered part of this mission, because we include these later in the discussion of defense suppression.

We can assume that the Soviet armored attack will have a network of rolling SAM defenses, providing a dense multimode, all-altitude defensive umbrella over the battlefield. The radars will be blinking (to avoid detection), and communications traffic will be heavy.

Conventional EW techniques include active radar and communications jamming by aircraft such as the EA-6B and the EF-111, laying of chaff along key air corridors, and standoff jamming by aircraft and ground units. Cooperative blinking of the airborne jamming radars is often necessary for survival, as is maintenance of special aircraft formations (Henderson, 1982).

RPVs may be used in this environment in a variety of ways. The Aquila and Scout have both been designed with onboard and expendable jammer packages (Hoisington, 1984). The RPVs' small signature allows them to penetrate and place jamming sources near the threat rather than utilizing standoff systems which are degraded by distance, weather, and terrain. Onboard jamming of enemy search and track radars also benefits from short ranges. Low power over the target is more effective than standoff jamming and does not deny portions of the spectrum to friendly forces. For communications jamming, RPVs can use the delay-and-retransmit (DART) technique. An elevated DART RPV in proximity to an enemy line-of-sight transmitter could intercept transmissions and retransmit them after a short delay. This retransmission effectively jams the link while relaying transmissions in a clear, unjammed form to intercept operators on the friendly side of the FLOT. In all these functions, frequent inter-RPV coordination will be necessary. The RPVs may need to negotiate over targets, so that several do not jam the same one. Or several may have to jam the same target from different directions. The jamming RPVs may have to blink synchronously to avoid antijamming missiles. Downed RPVs assigned to high-value targets may have to be replaced by others having lower-valued targets.

The primary measures of effectiveness in the EW mission would be the volume of message traffic jammed and enemy sensing opportunities missed. Indirect measures should include reduced accuracy and timeliness of enemy situation assessments, relative losses, and FLOT movement.

Communications Relay

Communications relaying is probably the simplest of the four RPV missions considered. It involves maintenance of a flexible network of RPVs able to maintain line-of-sight transmissions between command centers, aircraft, and ground units. The main problems are jamming, own radio-frequency interference (RFI) noise, bandwidth limitations, security, and losses.

Small communications repeater stations are now available for use on RPVs. One lightweight (10-lb) unit is able to receive and relay messages on separate frequencies of various signal structures, including voice, encrypted voice, and slow television scan within a 200-mi radius (Ingebretsen, 1982). Some more sophisticated techniques (e.g., communication beam steering, onboard signal processing and data compression, spread-spectrum techniques, and data-link protection) are now in development, although power, reliability, and cost are still major problems.

For our purposes, it would be useful to evaluate the performance of a group of RPVs with VHF communications stationed between non-line-of-sight units, such as surveillance aircraft, command centers, and artillery. The RPVs would have to coordinate positioning among themselves and reorganize due to attrition. They would also have to maneuver to avoid threats and minimize jamming effects. The primary performance measures would be communication throughput, transmission quality, and delays.

Defense Suppression

RPVs have been proposed for attack operations against radars, command centers, ground vehicles, helicopters, shipping, and even aircraft. The RPVs may carry an integral warhead, or they may launch missiles or bombs. The attacks may be performed independently or in concert.

Antiradar operations appear to be particularly important. Conventional use of passive-homing missiles such as Shrike or Martel can shut down the enemy radars only for the duration of the attack, but large numbers of low-cost harassment drones might be able to impose a virtual radar blackout (Hyman, 1981, Correll, 1984). The RPVs would circle over the battlefield, diving to engage any radars that began transmitting. Should an RPV be detected and the radar switched off, the RPV would climb back into its parking orbit. Both passive sensing of radar emissions and active sensing of the large GCI dish are possible with RPVs, but some tradeoff of sensing equipment and explosive payload must be made. Coordination among RPVs may be needed to assign attackers to targets, to perform assessment and retargeting, and to reorganize following losses.

Antiaircraft operation is a special case. It is unlikely that an RPV can down an aircraft in the air, but it should be able to identify and attack an aircraft taking off from a runway. Like the antiradar function described above, this capability should help pin down the air defense during an attack. For simplicity, we should probably concentrate on suppression of ground-based air defenses. The primary performance measures include the number of enemy assets damaged and destroyed and the number of own aircraft lost.

RESEARCH DIRECTIONS

These four missions provide an excellent application area for development and demonstration of DAI techniques. Taken together, the missions involve many of the key cooperative behaviors we are interested in: task distribution, communication management, situation assessment, cooperative planning, and human interface.

We are currently using the first mission, surveillance, in our ROSS-based multiple-RPV simulation (described in detail in Sec. III). The primary behaviors here are role negotiation, communication management, and data fusion. Our work with this simulation has shown the importance of the role-negotiation process in cooperative organizations and has demonstrated the efficacy of the ROSS language for simulation of such distributed military systems. We have also found that distributed situation assessment and planning are advantageous in certain surveillance situations.

The EW mission should be particularly good for studying opponent modeling and synchronization of actions. The RPV group may need to model the decision processes of the defenses to decide when and how to jam. This can be especially important during deception. Synchronization of actions occurs during coordinated blinking of jammers to avoid detection and loss, and during cooperative jamming of the same target. We have examined some aspects of action synchronization during our studies of formation flight and communication management, but the more complex behaviors needed in EW operations should be much more demanding and illuminating.

The third mission, communications relaying, is one in which network position is paramount (for maximizing bandwidth and minimizing noise, security problems, and vulnerability). This can be considered a more complex form of patterned flight, exercising the functions of coordinated execution and monitoring. In our simulations, we have demonstrated how an RPV group flying in formation can respond to contacts and threats by changing formation type, spacing, and leader position. The task of maintaining special geometries for relaying (much like optimizing a satellite network configuration) is a more difficult and general problem than is formation flight.

The final mission, defense suppression, is one in which the RPVs have many options; consequently, it involves deep planning. The options include gathering information, loitering, decoying, attacking defenses or targets, and assessing damage. Analysis of these options requires modeling of the opponent and stringing together many sequential actions. RPVs will have to evaluate each resulting plan over many moves and negotiate over plans and roles. Even with abstract scenarios, our studies have demonstrated the importance of deep, simulation-based planning rather than simple condition matching of actions.

Our research has concentrated on the surveillance mission, but aspects of all four key missions are included in our demonstration systems. Future work should expand the demonstration system to include operations sequencing through all four missions, exercising behaviors for the individual missions themselves as well as for mission transitions.

Appendix B

HOSES: A MECHANISM FOR INTERMODULE COMMUNICATION IN A MULTILANGUAGE, MULTIMACHINE ENVIRONMENT

We have explored the development of a multiprocessor system for implementing the surveillance RPV simulation. Ideally, the system should use different processors to run the simulation, the graphics, and certain database functions. This will be a multiple-language environment, with programs running as different processes on different machines connected by a network. In this appendix, we describe a method we developed for making these connections. We begin by noting several important software architecture issues.

First, the architecture itself must be highly flexible. For example, the logical boundary between the simulation per se and its graphic output processing may be implemented either within a processor or across a network. Once a logical piece of the processing has been split off to run as a separate process, it should be transparent to the rest of the code, whether it is on the same machine or another one.

It is also important to facilitate writing different pieces of code in different languages, both to take advantage of specific features of a language and to allow use of existing code. For example, some of the graphic processing, which was originally written in C, was rewritten in LISP; with the switch to GKS graphics, C may again be needed to provide interface code.

Finally, when the simulation itself is split into multiple processes (whether for enhanced performance through parallelism or for modeling realism), the ROSS message-passing mechanism will have to be extended to allow objects to be separated by process (or machine) boundaries. If modeling realism is the goal, it may be desirable to maintain an explicit distinction between passing messages among objects in the same process and passing messages across process boundaries; on the other hand, if enhanced performance is the goal, then the distinction should be minimized. In either case, some new forms of interprocess communication are needed.

The common thread in all of these activities is the need for a transparent and efficient method of encapsulating intermodule interfaces without requiring the programmer to decide in advance which of several alternative implementations will be used. We have designed a prototype mechanism for such encapsulation, which we call a "hose" (because it is similar to a UNIX "pipe" but more flexible). This is an attempt to build a fairly general intermodule call facility which can be used by a number of languages (initially C and LISP) for interprocess (or intraprocess) communication on the same machine or across a network.

We initially programmed a ROSS simulation of a hose facility as a high-level specification. This simulation contains objects corresponding to modules, processes, and machines. Each machine has a special net-server process which communicates with its counterparts on other machines. There is also a special hose-interface object for each machine, which represents the protocol that specifies interfaces between languages, processes, and machines. Each process belongs to some machine, and each module belongs to some process. When module P needs to communicate with module Q, it calls (sends a message to) the hose interface on its machine. The hose interface decides whether module Q is in the same process as module P, and if not, if it is on the same machine; in making this decision, the hose interface behaves as an active ROSS object (although its activity represents the functioning of a static piece of interface code). Intraprocess calls are passed directly from module P to module Q as ROSS messages. Interprocess calls cause an instance of a hose object to be created. If P and Q reside on the same machine, the hose performs the interprocess communication directly. Otherwise it sends a message to the local net-server, which communicates with its counterpart on the appropriate machine; the counterpart net-server creates a hose to deliver the message from P to Q. Acknowledgments are produced whenever hose objects are created. In the simulation, instances of hoses exist only long enough to perform a single interprocess call; they disappear when they receive acknowledgment of delivery.

An intermodule call using a hose would have the form of a procedure call with parameters specifying the procedure to be called and the values (and possibly the types) of the parameters to be passed to it. The exact form of a hose call in a given language is determined by a hose interface written for that language. For example, in Pascal, such a call might look like:

```
HOSE(ModuleA, ProcedureQ, Param1, PType1, Param2, PType2 . . .)
```

where ModuleA names a module that may reside on any machine, ProcedureQ names the procedure to be called in ModuleA, and each pair

Parami, PTypei is the value and type of a parameter to be passed to ProcedureQ. In order to be useful, the hose must allow typed parameter values rather than simple streams (whether ASCII characters or raw bits). The hose must support a range of "hose-defined" types including the common types in most programming languages. Each PTypei in the above example is therefore the hose-defined type of the corresponding Pascal value Parami. In a language where the type of an expression can be determined at run time, the user would not need to specify hose-defined types explicitly in the "hosecall" (the calling of a function via the hose), since the hose interface could supply the types itself.

An initial hose facility has been implemented (in C and LISP) and has been used to implement GIL. This hose facility allows a LISP program to call a C function that has been loaded into the LISP process space. The two LISP dialects supported so far are Franzlisp and PSL (including RLISP85, which translates directly into PSL). The Franzlisp version of hoses works on both the Sun workstation and the VAX (running UNIX 4.2 BSD).

The hose facility currently allows LISP programs to call C functions of up to 20 words of argument (e.g., 20 integers or 10 double floats), including pointers to arbitrarily large strings and arrays. (This 20-word limit is not an absolute upper bound; it can be extended if required.) A called C function can return single- or double-word results and can modify "out" parameters passed as pointers (in typical C style). It is also possible for a called C function to return a pointer to a structure that it has allocated, although this must be done carefully to ensure that LISP never tries to garbage-collect the C program's structure.

The primary intent of the hose facility is to allow LISP programs to use existing C function libraries without having to modify the C source code. This goal has essentially been met: Called C functions are completely unaware that they are being called by LISP. In cases requiring special manipulation of data, a "C-wrapper" function can be written which is hosecalled by LISP and which calls the target C function after performing any necessary transformations; in practice, this is rarely necessary.

The types currently supported include integers, strings, floating point numbers, pointers, and arrays of any of the above. In addition, C functions of a variable number of arguments (up to a declared maximum) can be called. The GIL facility uses hoses to call the existing GKS or Core library packages (compiled from C) in a clean and flexible way; the LISP (ROSS) programmer is shielded from the details of the C calling discipline and is able to treat the hosecalled graphics functions as extended primitives in LISP.

Appendix C

PROLOG: DESCRIPTION AND COMPARISON WITH LISP

For certain problems in our project, we found Prolog a more appropriate programming structure than LISP or ROSS. We implemented Prolog in LISP so that programs in it could interface with the rest of the LISP and ROSS code. In this appendix, we describe Prolog, show how it was used in our project, and compare it with LISP. We assume that the reader has basic familiarity with LISP.

PROLOG

Prolog is a language that can be used for logic programming, i.e., programming in which computation can be regarded as deduction. A logic program consists of a set of statements or clauses of the form:

$$A \leftarrow B1, \ldots, Bk \qquad k \geq 0$$

where each of A, Bi is a condition. The clause is read, "A holds if each of Bi hold." A condition is of the form R(t1, . . . , tn), $n \geq 0$, where R is an n-ary relation symbol, and each ti is a term. A term is either a variable, a constant, or a function application of the form f(x1, . . . , xm), $m \geq 0$, where f is an m-ary function symbol, and each xi is a term. All variables in this clause are *universally* quantified upon.

The set of clauses of the form R(t1, t2, . . . , tn) $\leftarrow$ B can be thought of as a definition of relation R. As shown below, this definition can be used to compute which n-tuples of terms are in relation R. Note that all clauses are first-order, i.e., functional and relation symbols are not quantified upon.

Given a logic program S, a query upon S is a conjunction Q of conditions:

B1, B2, . . . , Bk.

Each variable in Q is existentially quantified. If x1, . . . , xn are the variables in Q, then the problem is to show, or prove, that there exist x1, . . . , xn such that B is a logical consequence of S. If the proof is found, then, indirectly, values of x1, . . . , xn are computed.

The inference procedure used to find the above proof, if it exists, is called SLD-resolution (Kowalski, 1974; van Emden, 1977). This procedure works in a top-down manner. To prove a conjunction of conditions Q1, . . . , Qi − 1, Qi, Qi + 1, . . . , Qn from a set of clauses S, it selects a condition Qi and a clause A ← B1, . . . , Bk in S such that A and Qi match with some most general substitution α. It then attempts to prove a new query, which is the result of instantiating

Q1, . . . , Qi − 1, B1, . . . , Bk, Qi + 1, . . . , Qn

with α. The procedure halts when the query is empty, i.e., when there is nothing more to prove.

In the above procedure, a special form of matching, called unification, is used. Two terms A and B unify if there is some substitution s such that A and B when instantiated with s yield the same term. For example, the terms f(X, X, 2) and f(3, 3, Y) unify with substitution {<X,3>, <Y,2>}, but the terms f(X, X) and f(2, 3) do not unify.

SLD-resolution satisfies the soundness and completeness properties. Soundness means that any conclusion drawn by the procedure is correct; completeness means that if a query is a logical consequence of some logic program, it will be proved to be so in finite time. Roughly, completeness guarantees that if an answer exists, it will be found in finite time.

Prolog is an implementation of SLD-resolution, augmented with common programming utilities.[1] Current Prologs are as fast as LISP (Warren et al., 1977), and they surpass LISP for performing tasks requiring inference. Whereas these tasks can be programmed directly in Prolog, with LISP we would first have to program an inference procedure and then program the task.

Since its appearance about a decade ago (Warren et al., 1977), Prolog has been applied with tremendous success to many areas of computer science and artificial intelligence, including programming languages, databases, natural language analysis, rule-based reasoning, meta-level reasoning, constraint satisfaction, pattern matching, symbolic algebra, distributed processing, and search. More important, the logical interpretation of Prolog has provided very useful *formalizations* of ideas in these areas (Clark and Tarnlund, 1982; van Caneghem and Warren, 1986).

[1]This is not entirely correct. To make SLD-resolution practical on a conventional computer, some compromises have been made in the design of Prolog. However, over a wide range of applications, Prolog remains almost identical to SLD-resolution.

We considered Prolog to be more appropriate than LISP for two problems in the RPV project: communication relaying and data inference. The first problem was to find the shortest communication path between two RPVs in a communication network that consists of point-to-point links between each pair of RPVs. Since links may sometimes be electronically jammed, a message may have to be relayed by other RPVs before it reaches its destination. The problem is to find the shortest relay path.

There is a well-known algorithm for solving this problem. However, the Prolog version, though less efficient, is simpler and much more flexible. An open path between two RPVs is a sequence of RPVs connected to each other by open links. We can express this in Prolog as:

```
open_path(X,X,[X]) ←

open_path(X,Y,[X|R]) ← open_link(X,Z),open_path(Z,Y,R).
```

The first clause states that the path from X to itself is a sequence containing just X. The second clause states that a path from X to Y is the sequence [X|R] such that there is an open link between X and an interim RPV Z, and R is the path between Z and Y. At any given time in the simulation, the state of the network is modeled by a set of clauses of the form open_link(A,B) or jammed_link(A,B) where A and B represent RPVs. For example, we can have:

```
open_link(RPV1, RPV2)

open_link(RPV2, RPV3)

open_link(RPV3, RPV4)

open_link(RPV2, RPV4)

jammed_link(RPV1, RPV4)
```

The definition of open_path can now be used by Prolog in several different ways.

1. Given two RPVs and an alleged open path between them, Prolog can determine whether the path satisfies the above definition. For example, it can show that

```
open_path(RPV1, RPV4, [RPV1, RPV2, RPV4])
```

is true.

2. Given two RPVs, Prolog can find, *without further programming*, all the open paths between them. The query

```
setof(P,open_path(RPV1,RPV4,P),S)
```

will bind S to

```
[[RPV1, RPV2, RPV4], [RPV1, RPV2, RPV3, RPV4]].
```

The shortest path can now easily be found.

3. Given a path P and the first RPV, say R1, Prolog can determine what the second RPV, R2, must be such that P is an open path between R1 and R2. Of course, R1 can be determined similarly when R2 and P are known. Thus, while the query

```
open_path(RPV1, R, [RPV1, RPV3, RPV4])
```

will fail, the query

```
open_path(RPV1, R, [RPV1, RPV2, RPV3, RPV4])
```

will succeed with R bound to RPV4.

4. Given a path P, Prolog can determine the pair of RPVs such that P is an open path between them. Thus, the query

```
open_path(R1, R2, [RPV2, RPV3, RPV4])
```

will bind R1 to RPV2 and R2 to RPV4.

We emphasize that the above possibilities are quite natural when we consider the *logical interpretation* of Prolog. In all cases we have a query of the form open_path(X1, X2, X3) in which some Xi may be variables. The problem is to find a value of these variables such that the query, when instantiated with these values, is a logical consequence of the open_path program. Logic is neutral as to which of the Xi are variables. That is, it does not maintain any distinction between input and output variables. Any of the Xi can be input or output. In contrast, the usual shortest-path algorithm can be used in only one way, and that is to find the shortest path.

The second problem for which we found Prolog particularly appropriate concerns decisionmaking in a dangerous environment. Often an RPV requires some information that another RPV possesses.

The first RPV could explicitly request the information, but this could be dangerous, especially when enemy ground sensors may be listening. Alternatively, the RPV could attempt to infer the information, based upon its current view of the world. Inference can, however, be time-consuming. Moreover, if the RPV's view of the world is incorrect, its conclusions could also be incorrect.

Our RPVs follow the simple strategy of first attempting to infer the information; if sufficient data are not available, they then explicitly request information from whichever RPV has it. Rules for inferring information are, of course, readily expressed in Prolog. For example, one rule is that if a SAM fires on an RPV, the SAM-to-RPV distance is greater than 5 units, and no other known defenses are in the area, assume the radar is in the area. This rule in Prolog is:

```
radar_in_area(R1) ← known_defenses(R1, [ ]),
                    fires(SAM, R2,T),
                    within(5.0 R1 SAM).
```

For an RPV, R, known_defense(R, L) is true when L is the list of defenses R knows about. L can of course also be the empty list [ ]. Whenever a SAM S fires on an RPV R at time T, a clause fires(S, R, T) is added to the history of the simulation. Condition within(D, R, S) is true when RPV R is within distance D of SAM S.

An RPV can use this rule to determine whether it should assume that it is near a radar. However, the user can use the same rule to determine which RPVs could assume this hypothesis. In a similar manner, five other rules involving various complex combinations of conditions were coded in Prolog.

COMPARING PROLOG WITH LISP

LISP supports the functional style of programming (Abelson and Sussman, 1984). A LISP program can be regarded as a set of definitions of functions from symbolic expressions to symbolic expressions. Given as input the arguments of that function, LISP will deliver as output the result of applying that function to those arguments. That is, LISP computes functions, whereas Prolog computes relations. Since a relation is a more general concept than a function, Prolog can be thought of as being more general than LISP.

This generality is exemplified by the directness with which Prolog can be used for programming problems in many areas of artificial intelligence, e.g., databases, natural-language analysis, rule-based reasoning. Many ideas in artificial intelligence are captured more naturally in terms of relations (and connections between them), than in terms of functions.

In particular, a LISP program cannot be used in the flexible ways (described above) that an equivalent Prolog program can. We would have to write a separate LISP program for each of these uses. LISP's basis in functional programming forces it to maintain a well-defined distinction between input and output. Moreover, LISP could not be used to directly program rules of inference in a dangerous environment, simply because LISP does not support the notion of a rule.

The presence of unification pattern matching in Prolog considerably increases program clarity and conciseness. For example, the Prolog and LISP programs for appending two lists are, respectively,

```
append([ ], X, X).

append([U|V], W, [U|Z]):-append(V, W, Z).

(defun append (x y)

    (if (null x) then y else (cons (car x) (append (cdr x) y))))
```

Note the absence of selector functions in the Prolog version. A list can be specified either as [ ] or as [U|V], where [ ] is the empty list and [U|V] is a nonempty list whose head is U and whose tail is V.

However, there is an important respect in which LISP is more powerful than Prolog: It supports higher-order functions, i.e., functions that either take functions as input or return functions as output. Higher-order functions can be used to implement sophisticated data and control structures, e.g., object-oriented programming or demand-driven computation.

For example, iteration can be very conveniently expressed using the higher-order function map

```
(define (map f l) (cond ((null l) nil)
                        (t (cons (f (car l)) (map f (cdr l))))))
```

That is, (map f l) takes a function f and a list l as arguments, applies f to every element of l, and returns the list of the results. For example, (map add1 '(1 2 3 4)) will evaluate to (2 3 4 5). In the same way, (map (lambda (x) (times x x)) '(1 2 3 4)) will evaluate to (1 4 9 16). Thus, map works for any function and list.

Another example of the use of higher order functions is in demand-driven processing, where an expression is evaluated only when there is demand for its value. The mechanism for suspending evaluation is easily implemented using a higher-order function.

For example, we define the function intfrom to take a natural number N and return the list of all natural numbers starting at N. With the obvious definition,

```
(define (intfrom N) (cons N (intfrom (+ N 1))))
```

the expression (intfrom 2) will cause a nonterminating computation. However, with an alternative definition,

```
(define (intfrom N) (lambda (S) (cond ((eq S 'head) N)
                                       (t (intfrom (+ N 1))))))
```

the expression (intfrom 2) will yield a function F representing the *suspended* computation of the list of natural numbers starting at 2. We can demand that computation proceed by simply calling F. To find the first element of F, we do (F 'head); to find the second element, we do ((F 'tail) 'head); and so on.

Because Prolog is a first-order language, we cannot use it directly for higher-order programming. That is, we cannot treat relations as "first-class" objects. In particular, we cannot define a single equivalent of map that would work for all relations. We also cannot do demand-driven processing in *exactly* the way described above.[2]

CONCLUSIONS

In spite of the above differences, Prolog and LISP are in a class quite apart from other procedural languages, such as Fortran, Pascal, or C. Their similarity derives from their common basis in logic.

Logic is an old subject. It consists of posing and settling fundamental questions in fields such as mathematics, language, philosophy, or computing. Over its long history, it has developed a large collection of interesting and powerful concepts. Because of their fundamental nature, these concepts can, and often do, provide useful interpretations of knowledge in many other fields. Exploring such interpretations,

[2]However, there is another way to do demand-driven processing in Prolog which is equally general, natural, and efficient (Narain, 1985).

particularly in unformalized domains such as of military strategies and tactics, is a very fruitful line of research.

The greatest importance of Prolog and LISP is that they make computationally practical some important concepts in logic, in particular, in the lambda calculus and the predicate calculus. (The former is a formalized language of functions while the latter is a formalized language of relations.)

Prolog and LISP can thus be used for representing knowledge in unformalized fields and for providing a formal interpretation for such knowledge. They can also be employed for making computationally practical other useful, but more complex concepts in logic. Thus a programming environment in which both LISP and Prolog are resident holds considerable promise for AI research.

REFERENCES

Abelson, H., and G. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press and McGraw-Hill, New York, 1984.

Appelt, D. E., *Planning Natural-Language Utterances to Satisfy Multiple Goals*, Technical Note 259, SRI International, 1982.

Caldwell, H., and F. Kennedy, Jr., "Stepchild of Unmanned Vehicles," *National Defense*, Vol. LXVII(380), September 1982, pp. 16–20, 31–32.

Cammarata, S., D. McArthur, and R. Steeb, *Strategies of Cooperation in Distributed Problem Solving*, The Rand Corporation, N-2031-ARPA, October 1983.

Carbonell, J., "POLITICS: Automated Ideological Reasoning," *Cognitive Science*, Vol. 2, 1978, pp. 27–51.

Clark, K., and S. Tarnlund, *Logic Programming*, Academic Press, New York, 1982.

Correll, J., "Where TAC AIR Is Heading," *Air Force Magazine*, Vol. 67, No. 6, June 1984, pp. 50–58.

Dahl, O-J., and K. Nygaard, "Simula—An Algol-Based Simulation Language," *Communications ACM*, Vol. 9, 1966, pp. 671–678.

Dalkey, N. C., *Group Decision Making*, Report UCLA-ENG-7749, School of Applied Science, University of California, Los Angeles, July 1977.

Davis, R., *A Model for Planning in a Multi-Agent Environment: Steps Toward Principles for Teamwork*, Working Paper, MIT Artificial Intelligence Laboratory, 1981.

Davis, R., and R. G. Smith, *Negotiation as a Metaphor for Distributed Problem Solving*, Memo 624, MIT Artificial Intelligence Laboratory, 1981.

Duda, R. O., J. G. Gaschnig, and P. E. Hart, "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in D. Michie (ed.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, Edinburgh, 1979, pp. 153–167.

Ellis, J. W., Jr., *A Role for Remotely Manned Sensors over the Battlefield*, the Rand Corporation, P-6255, December 1978.

Erman, L. D., P. E. London, and S. F. Fikas, "The Design and an Example Use of HEARSAY-III," *IJCAI*, Vol. 7, 1981, pp. 409–415.

Fahlman, S., "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, Vol. 5, No. 1, 1974, pp. 1–49.

Fikes, R. E., and N. J. Nilsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, No. 2, 1971, pp. 189–208.

Forgy, C. L., *The OPS5 Users Manual*, Technical Report CMU-CS-79-132, Computer Science Department, Carnegie-Mellon University, Pittsburg, 1981.

Garvey, T., J. Lowrance, and M. Fischler, "An Inference Technique for Integrating Knowledge from Disparate Sources," *IJCAI*, Vol. 7, 1981, pp. 319–325.

Goldberg, A., and A. Kay, *Smalltalk-72 Instruction Manual*, Report SSL 76-6, Xerox PARC, Palo Alto, 1976.

Gordon, J., and E. Shortliffe, "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space," *Artificial Intelligence*, Vol. 26, 1985, 323–357.

Gossett, T., and F. Velligan, "The Aquila: A Versatile, Cost-Effective Military Tool Shows Its Potential," *Military Electronics/Countermeasures*, Vol. 8, No. 12, December 1982, pp. 74–78.

Gray, C., K. Rehm, and D. Woods, "Drone Formation Control System," *Military Science*, 1982, pp. 11–26.

Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, Vol. 26, 1985, pp. 251–321.

Henderson, C., "Sea-Based Remotely Piloted Vehicles," *Military Electronics/Countermeasures*, Vol. 8, No. 5, May 1982, pp. 45–47.

Hoisington, D., "Short Course on Electronic Warfare," presented at The Rand Corporation, July 1984.

Hyman, A., "Where Are the RPVs?" *Aerospace International*, Vol. 17, No. 3, July/August 1981, pp. 40–43.

Ingebretsen, D., *E-Systems Corporation, RPV Brochure*, Melpar Division, 1982.

Jane's All the World's Aircraft, Section on RPVs and Targets, 1983–84.

Jefferson, D., and H. Sowizral, *Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control*, The Rand Corporation, N-1906-AF, December 1982.

Kahn, R., S. Gronemeyer, J. Burchfiel, and R. Kunzelman, "Advances in Packet Radio Technology," *Proceedings of the IEEE*, Vol. 66, No. 11, November 1978, pp. 1468–1496.

Kiviat, P., R. Villanueva, and H. Markowitz, *The Simscript II Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1968.

Klahr, P, D. McArthur, S. Narain, and E. Best, *SWIRL: Simulating Warfare in the ROSS Language*, The Rand Corporation, N-1885-AF, September 1982.

Klahr, P., J. W. Ellis, Jr., W. Giarla, S. Narain, E. Cesar, and S. Turner, *TWIRL: Tactical Warfare in the ROSS Language*, The Rand Corporation, R-3158-AF, October 1984.

Klass, P., "Multi-Drone Control Unit in Development," *Aviation Week and Space Technology*, February 10, 1975, pp. 57–59.

Klass, P., "Lebanon Lessons Raise Interest in RPVs," *Aviation Week and Space Technology*, Vol. 121, No. 8, August 20, 1984, pp. 44–46.

Konolige, K., "A First Order Formalization of Knowledge and Action for a Multi-Agent Planning System," *Machine Intelligence*, Vol. 10, 1981.

Konolige, K., and N. Nilsson, *Multiple Agent Planning Systems*, Artificial Intelligence Center Report, SRI International, April 30, 1980.

Kowalski, R., "Predicate Logic as a Programming Language," *Proceedings of the IFIP Congress*, 1974.

Kowalski, R., *Logic for Problem Solving*, North Holland, 1979.

Kunz, J. C., R. J. Fallat, D. H. McClung, J. J. Osborne, R. A. Votteri, H. P. Nii, J. S. Aikins, L. M. Fagan, and E. A. Feigenbaum, *A Physiological Rule-Based System for Interpreting Pulmonary Function Test Results*, Report HPP-78-19, Heuristic Programming Project, Computer Science Department, Stanford University, Stanford, Calif., 1978.

Lesser, V., *A High-Level Simulation Testbed for Cooperative Problem Solving*, COINS Technical Report 81-16, University of Massachusetts, Amherst, 1981.

Lesser, V., and D. Corkhill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks," *The AI Magazine*, Vol. 4, No. 3, Fall 1983, pp. 15–33.

Lesser, V., S. Reed, and J. Pavlin, "Quantifying and Simulating the Behavior of Knowledge-Based Interpretation Systems," *Proceedings of the First Annual National Conference on Artificial Intelligence*, Stanford University, Stanford, Calif., 1980.

Lozano-Perez, T., "Robot Programming", *Proceedings of the IEEE*, Vol. 71, No. 7, July 1983, pp. 821–841.

Lupo, J., "Tactical Autonomous Weapon Systems," *Unmanned Systems*, Vol. 2, No. 4, Spring 1984, pp. 7–9.

McArthur, D., and P. Klahr, *The ROSS Language Manual*, The Rand Corporation, N-1854-1-AF, September 1985.

McArthur, D., P. Klahr, and S. Narain, *ROSS: An Object-Oriented Language for Constructing Simulations*, The Rand Corporation, R-3160-AF, December 1984.

McArthur, D., R. Steeb, and S. Cammarata, "A Framework for Distributed Problem Solving," *Proceedings of the National Conference on Artificial Intelligence*, Pittsburg, 1982, pp. 181–184.

Narain, S., "A Technique for Doing Lazy Evaluation in Logic," *Proceedings of the Second IEEE International Symposium on Logic Programming*, Boston, July 1985.

Newell, A., and H. Simon, "GPS-A Program That Simulates Human Thought," in E. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.

Newell, A., and H. Simon, *Human Problem Solving*, Prentice-Hall, New York, 1972.

Nii, H. P. and N. Aiello, "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs," *IJCAI*, Vol. 6, 1979, pp. 645–655.

Quinlin, R., *Inferno: A Cautious Approach to Uncertain Inference*, The Rand Corporation, N-1898-RC, September 1982.

Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol. 5, No. 2, 1974, pp. 115–135.

Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier North-Holland, New York, 1977.

Sanders, J., "World Without Man," *Defense and Foreign Affairs*, Paris Air Show Edition, 1981, pp. 29–32.

Shafer, G., and A. Tversky, "Languages and Designs for Probability Judgement," *Cognitive Science*, Vol. 9, 1985, pp. 309–339.

Shortliffe, E.H., *Computer-Based Medical Consultation: MYCIN*, American Elsevier, New York, 1976.

Singh, V., and M Genesereth, *A Variable Supply Model for Distributing Deductions*, Report HPP-84-14, Heuristic Programming Project, Computer Sciences Department, Stanford University, Stanford, Calif., May 1984

Smith, B. A., "Israeli Use Bolsters Interest in Mini-RPVs," *Aviation Week and Space Technology*, July 18, 1983, pp. 67–71.

Smith, R. G., *A Framework for Problem Solving in a Distributed Processing Environment*, STAN-CS-78-700, Stanford University, Stanford, Calif., 1978.

Steeb, R., and J. Gillogly, *Design for an Advanced Red Agent for the Rand Strategy Assessment Center*, The Rand Corporation, R-2977-DNA, May 1983.

Steeb, R., S. Cammarata, S. Narain, and W. Giarla, *Distributed Problem Solving for Air Fleet Control: Framework and Implementations*, The Rand Corporation, N-2139-ARPA, April 1984.

Steeb, R., S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke, and R. B. Wesson, *Distributed Intelligence for Air Fleet Control*, The Rand Corporation, R-2728-ARPA, 1981.

Stefik, M., "Planning with Constraints (MOLGEN: part 2)," *Artificial Intelligence*, Vol. 16, 1981, pp. 141–169.

Sussman, G., *A Computational Model of Skill Acquisition*, American Elsevier, New York, 1975.

van Carneghem, M., and D. Warren, *Logic Programming and its Applications*, Ablex Publishing, 1986.

Van Emden, M., *Programming in Resolution Logic*, Machine Intelligence, Vol. 8, 1977.

Van Melle, W., "A Domain-Independent Production-Rule System for Consultation Programs," *IJCAI*, Vol. 6, 1979, pp. 923–925.

Warren, D., L. Periera, and F. Periera, *Prolog—The Language and its Implementation Compared to LISP*, SIGPLAN Notices, Vol. 12, No. 8, and SIGART Newsletter 64, August 1977.