

AFRL-IF-RS-TR-2003-33
Interim Technical Report
February 2003



DYNAMIC CONTROL AND FORMAL MODELS OF MULTI-AGENT INTERACTIONS AND BEHAVIORS

ALPHATECH, Incorporated

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K542

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

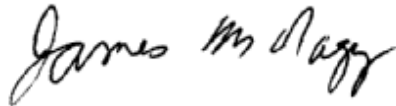
The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

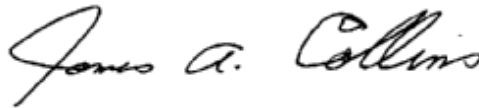
AFRL-IF-RS-TR-2003-33 has been reviewed and is approved for publication.

APPROVED:



JAMES M. NAGY
Project Engineer

FOR THE DIRECTOR:



JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE FEBRUARY 2003	3. REPORT TYPE AND DATES COVERED Interim Jun 00 – Oct 02	
4. TITLE AND SUBTITLE DYNAMIC CONTROL AND FORMAL MODELS OF MULTI-AGENT INTERACTIONS AND BEHAVIORS			5. FUNDING NUMBERS C - F30602-00-C-0182 PE - 62301E PR - TASK TA - 00 WU - 01	
6. AUTHOR(S) Jeff Coble, Larry Roszman, and Tiffany Frazier				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ALPHATECH, Inc. 3811 North Fairfax Drive, Suite 500 Arlington Virginia 22203			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/ITB 3701 North Fairfax Drive Arlington Virginia 22203-1714			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2003-33	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: James M. Nagy/ITB/(315) 330-3173/James.Nagy@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) New Multi-Agent System (MAS) approaches to complex DoD problems hold the promise of previously unrealized levels of autonomy, adaptability, and flexibility of agent-controlled systems. These systems will provide essential capabilities in command and control, surveillance, automated targeting and weapons delivery, and biochem monitoring. ALPHATECH's work has been focused on three areas. First is the development of an Open Experimentation Framework to facilitate research, evaluation, and characterization of the emerging science of Multi-Agent Systems. Second is the design and development of the Testbed for Taskable Agent Systems (TTAS), which is a software environment facilitating experimentation with disparate agent technologies and evaluation of critical design elements of Multi-Agent Systems. Lastly, our theoretical research developing cooperative methods for machine learning in Multi-Agent Systems and designing goal-directed agents that make and adapt decisions in a heterogeneous dynamic environment within a coherent mathematical framework of dynamic programming and Partially Observable Markov Decision Processes.				
14. SUBJECT TERMS Multi-Agent Systems, Agent-Based Computing				15. NUMBER OF PAGES 69
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

TABLE OF CONTENTS

1	SUMMARY	1
1.1	Objectives and Accomplishments.....	1
1.1.1	Open Experimentation Framework (OEF).....	2
1.1.2	Testbed for Taskable Agent Systems (TTAS).....	3
1.1.3	MAS Research	3
2	Introduction.....	4
2.1	Needs for Agent Technology	4
2.2	Barriers to Deployment of Multi-Agent Systems.....	5
2.3	TASK Roadmap.....	6
3	OEF.....	6
3.1	OEF History	8
3.1.1	IVRS	8
3.1.2	CAHDE.....	11
3.2	Design Problems.....	12
3.2.1	The General TASK OEF Design Problem and Experiment	13
3.2.2	The UAV Domain UAV-S (1) Baseline Problem and Experiment	15
3.3	Milestones Achieved and Future Direction	22
4	TESTBED FOR TASKABLE AGENT SYSTEMS.....	22
4.1	Overview	22
4.2	Technical Description	24
4.2.1	Multi-Agent Development Kit.....	25
4.2.2	TTAS <i>main()</i> Method	28
4.2.3	TTAS GUI	28
4.2.4	TTAS Experiment Simulation Configurer.....	29
4.2.5	TTAS Simulation Component	29
4.2.7	Air Vehicles	30
4.2.8	Tasks	34
4.2.9	The TTAS Simulation Environment Component	35
4.2.10	TTAS Umpire Component.....	35

4.2.11	TTAS Data Recorder	35
4.2.12	Two Dimensional View	36
4.2.13	The Three Dimensional View	36
4.3	Milestones and Future Directions.....	37
4.3.1	Milestones	37
4.3.2	Future Directions	37
5	COOPERATIVE LEARNING AND DYNAMIC CONTROL MAS RESEARCH	37
5.1	Objective	39
5.2	Reinforcement Learning	39
5.2.1	Cooperative Learning.....	40
5.3	Quality of Knowledge	41
5.3.1	Quality Measurement.....	41
5.3.2	Experiment.....	46
5.3.3	Preliminary Analysis.....	47
5.4	Continuing Research and Future Direction	49
5.4.1	Sequential Decision Problems	49
5.4.2	Problem Parameters	50
5.4.3	PO-MDP Controller Formulation	52
5.4.4	Observations	54
5.4.5	Actions	57
5.4.6	Policy	57
5.5	Conclusions and Continuing Research	58
6	PROJECT CONCLUSIONS AND CONTINUING WORK	59
6.1	Open Experimentation Framework	59
6.2	Testbed for Taskable Agent Systems	60
6.3	Multi-Agent System Research	60
7	REFERENCES.....	62

LIST OF FIGURES

Figure 1. Agent characterization and agent behavior models.....	10
Figure 2. Testbed for Taskable Agent Systems design.....	26
Figure 3. The Aalaadin agent organization model.....	27
Figure 4. The MadKit architecture (from Ferber).....	28
Figure 5. The basic vehicle reference implementation outline.....	33
Figure 6. TTAS' two-dimensional display.....	36
Figure 7. Interaction between the agent and its environment in a Reinforcement Learning context.....	38
Figure 8. Tabular SARSA(λ) online learning and control algorithm.....	40
Figure 9. As the degrees of freedom approach infinity, the T distribution converges to the Standard Normal curve.	45
Figure 10. Cooperative SARSA(λ) online learning and control algorithm.....	46
Figure 11. Experiments were conducted on a 10X10 grid environment with two terminal states, with values 10 and 100 respectively.....	46
Figure 12. (a) Convergence of reward and # of steps executed by agent. (b) Attenuation of e value.....	47
Figure 13. Certainty measurements for optimal actions, at 100-episode sampling increments.....	48
Figure 14. Measures for all actions at 800th episode.....	48
Figure 15. Agents navigating through a physical space with known and unknown tasks. Agents have distinct task surveillance, task detection, agent communication, and agent detection ranges, respectively illustrated by the four rings surrounding the agent, starting from the innermost ring.....	51
Figure 16. A possible PO-MDP controller formulation of a TASK Agent.	53
Figure 17. . State transitions reflecting discrimination on the task density within the agent's area of interest.....	54
Figure 18. Agents maintain information about the entire environment, but limit decision making to include information about a limited area of interest.	55

LIST OF TABLES

Table 1 The TTAS APIs for Air Vehicles	30
Table 2. The TTAS APIs for Tasks	34
Table 3. Source and content of all raw information, from which the agent constructs a set of observations.	55

1 Summary

The objective of this effort is to extend the current scientific and mathematical foundations of agent-based computing by adding rigor to the engineering of agent-based systems and tools in support of the Taskable Agent Software Kit (TASK) program.

The scope of the research effort is to develop mathematically correct techniques for modeling and analyzing agent behaviors, agent design methods, and the design of agent creation tools. Tasks include:

- **OEF**
 - Formulate and coordinate OEF parameters and activities for the TASK program that capture all key mathematical and theoretical facets of multi-agent systems within the context of multiple UAVs operating to service multiple tasks.
 - Develop models to describe, predict, and evaluate the behaviors of multi-agent systems (MAS).
- **MAS Research**
 - Extend computer theoretic models of coordination and adaptation to model autonomous interactive processes. Investigate approaches to incorporate interactive models of computation into the design and specification of agents and MAS.
 - Develop strategies to observe/control behavior of agents and MAS behavior on-line (i.e. emergent behavior).
- **Testbed for Taskable Agent Systems**
 - Develop a common framework within which the expressive power of a MAS implementation of a command and control system can be demonstrated
 - Develop (large-scale) models of MAS, develop prototypes/models of developed methods and techniques, and evaluate the viability and effectiveness of these techniques in modeling, analyzing, observing, and controlling MAS behavior.

1.1 Objectives and Accomplishments

ALPHATECH's work on the TASK program has included efforts on three distinct tasks: The development and maintenance of a Research Exploration Framework and evolving it into an Open Experimentation Framework for all program participants; Design and development of a Testbed for Taskable Agent Systems, which will serve as a common simulation environment for evaluation of program research; Research and Development of new agent theories designed to facilitate coherent behavior in Multi-Agent Systems.

1.1.1 Open Experimentation Framework (OEF)

One of ALPHATECH's tasks was the development and maintenance of the Control and Adaptation in Heterogeneous Dynamic Environments (CAHDE) Research Exploration Framework (REF) for a subset of the Projects in the TASK Program. The CAHDE REF provides:

- A framework to investigate solutions in dynamic control and adaptation of large-scale multi-agent system (MAS) behavior,
- A means to analyze agent/multi-agent system designs with respect to their ability to predict and control emergent behavior, and
- A context in which to demonstrate applicability and utility of multi-disciplinary MAS research to solve challenging dynamic command and control problems.

The military context of the CAHDE REF is the control and adaptation of autonomous air vehicles in uncertain, dynamic, heterogeneous, and hostile environments. This includes the military problems of air traffic control, dynamic resource allocation, sensor grid management, cooperative control, and self-organizing, self-healing, and self-regulating large-scale systems. Research foci of the TASK projects within the CAHDE REF are: mathematical underpinnings of modern computer science, new essential algorithms for key DOD needs, agent designs with quantifiable performance, and methodologies and tools to design and analyze military C2 MAS. ALPHATECH's success with the CAHDE REF has now being transitioned to an Open Experimentation Framework (OEF), which is serving as a context for the evaluation and analysis for *all* agent research conducted in the TASK program.

The Open Experimentation Framework (OEF) is a parameterized problem, solution, and experiment space that provides a well-defined, standard context in which researchers can apply dissimilar approaches to the same difficult problem set. Systematic variations to baseline experiments, which capture aspects of the problems, provide measurements of the solution parameters achieved by the different research approaches with respect to the problem parameters. At a minimum, these experimentally derived mappings of problem space to solution space provide system designers with guidance in the selection of particular implementation approaches. At a maximum, researchers into Multi-Agent System (MAS) architectures and designs may notice trends in these mappings that they can exploit for "better" MAS theories.

REF/OEF accomplishments:

- Successfully defined and administered the CAHDE REF and coordinated the participation and research of multiple project PIs.
- Identified and refined seven major critical elements for MAS research and evaluation
- Developed detailed OEF problem specifications and metrics for Adaptation and Coordination critical elements
- Defined and specified baseline parameters for the UAV-S(1) surveillance problem.

- Engaged all project PIs on OEF definition, problem specification, and integration of their research

1.1.2 Testbed for Taskable Agent Systems (TTAS)

The Testbed for Taskable Agent Systems is a simulation of m collaborative, adaptive, and autonomous agent-driven UAVs that *survey* and/or *search* for n ground sites. TTAS accommodates multiple approaches and implementations to multi-agent autonomy, adaptation, and collaboration, which can be mixed. It simulates only those features that might significantly affect Multi-Agent System properties and includes the ability for detailed data collection into a relational database, three-dimensional physical representation, standard, reproducible experimental configurations, and limited two- and three- dimensional displays of physical movement.

TTAS is designed to serve the following purposes:

- Investigate and define UAV kinematics models, *task* definitions, and standard experimental conditions of the DARPA TASK OEF
- A standard configuration managed platform on which one can perform standard, well-defined experiments on different MAS design and implementation approaches
 - Systematic, configuration managed variation of problem parameters
 - Systematic, configuration managed measurements
 - Easily incorporate additional managed variations and measurements particular to specific Multi-Agent System approaches
- Provide an experimental basis for the comparison of different Multi-Agent System approaches—particularly with respect to autonomy, adaptation, and coordination—under a wide range of identical problem conditions

TTAS accomplishments:

- Developed a scaleable, multi-threaded software system with the following features:
 - Well defined APIs supporting multiple interactive agents with different underlying implementations
 - Fully 3-dimensional environment
 - Six degrees of freedom for UAV motion
 - Relational database for extensive experiment data logging
 - XML configuration file to adjust the many system parameters

1.1.3 MAS Research

ALPHATECH's work has been focused on developing agent-based systems that are capable of adapting to dynamic environments while continuing to execute rational actions. As an agent

encounters the various states of the environment in which it operates, reinforcement learning (RL) enables the agent to iteratively learn a policy for rational action selection. The continual nature of reinforcement learning allows the agent to adapt its policy as the environment changes. In a MAS, the individual agents are working together to achieve a common goal and therefore must consider each other as part of the environment. The agents must understand how the behavior of other agents influences their ability to achieve objectives, either explicitly through interaction or implicitly through observation. ALPHATECH has been working to extend reinforcement learning theories so that agents can learn to consider their peers as a dynamic aspect of the environment and to interact effectively to improve the overall system performance.

MAS research accomplishments:

- Conducted extensive research efforts focused on the development of new mechanisms for coordination and interaction of agents operating in a Multi-Agent System.
- Developed a mechanism for measuring quality of knowledge learned through Reinforcement Learning
- Conducted experiments to evaluate quality measurement theories and methodology
- Initiated design and development of a Partially Observable Markov Decision Process framework for an agent controller capable of operating within a multi-agent system environment.

2 Introduction

The eventual deployment of large-scale networks of cooperative autonomous vehicles offers tremendous promise for future DoD missions in the following areas.

- Intelligence: UAVs for radar, imaging, and surveillance for military missions abroad
- Targeting: Autonomous UAVS able to cooperatively search, detect, identify, and destroy targets.
- Homeland security: Ubiquitous monitoring of environments for biological/chemical/nuclear agents
- Electronic monitoring for COMINT and SIGINT
- Autonomous Robot Teams for search & rescue or hazardous materials cleanup
- Inexpensive, fault tolerant, redundant, massively distributed systems for battlefield command and control

2.1 Needs for Agent Technology

Advances must be made in mechanisms for control, adaptation, and coordination of Multi-Agent Systems in order to support complex applications required for the DoD.

Advances in autonomy (Control) will allow heterogeneous units to operate independently (local goals & missions) and yet cooperate effectively to achieve group goals.

- Problem addressed: flexible, “run-time” *distribution of control*
- UAV example: new and existing UAVs can be used to achieve critical missions through regulation of collective behavior *without* centralized design

Advances in adaptability will provide agents with the ability to recognize and respond to unanticipated mission and environment dynamics by learning new behaviors

- Problem addressed: system *design* incompleteness
- UAV example: respond to new threats (e.g. handheld surface-to-air weapons), environments (e.g. sand storms), missions, objectives

Advances in coordination will allow agents to communicate local information, goals, and intent to improve group performance.

- Problem addressed: *design* interoperability
- UAV example: assuring UAVs share the information (situation assessment, goals, etc) needed to assure achievement of group goals

2.2 Barriers to Deployment of Multi-Agent Systems

Finally, there are several barriers to deploying Multi-Agent Systems applications. These barriers coincide with the goals of the TASK program.

- Evaluating Multi-Agent System designs: The complexity of Multi-Agent Systems makes their performance particularly difficult to analyze. Consequently, it is equally difficult to determine the best system design for a MAS. New methods are needed to determine when a MAS design is “good” or “better” than another, or even what constitutes “good enough?” Some of the complexities and open questions are:
 - Designs are not against a fixed requirements specification
 - Mission needs and environmental dynamics change over time
 - What agent designs lead to more effective MAS?
 - How to compare and select agent subsystem designs
 - Which coordination mechanisms are suitable for specific problem configurations?
 - When should agents adapt and how should they adapt?
 - How much information should be shared between agents?
 - How can emergent behaviors be predicted and controlled?
 - Assuring performance in unanticipated circumstances
 - How to understand MAS performance boundaries and achieve performance guarantees?
 - Automating the design and analysis of agents and MAS

2.3 TASK Roadmap

A key TASK program objective is to ensure the effective employment of multi-agent technologies into future, critical DoD systems. To do this the TASK program must develop a scientific base for high-confidence Multi-Agent Systems operating in dynamic environments, resulting in the following products:

- Agent Design Methodologies: which establish core decision and control mechanisms for agents and define parameters for effective use.
- MAS Analysis Methodologies: which enable the prediction and control of emergent behavior in large scale systems.
- Rules of thumb for MAS design to achieve performance guarantees.
- A set of general metrics that can be used to realize effective MAS designs.
- TASK Toolkits: Software tools for the synthesis and analysis of MAS
- Evaluation and transition of technological advances to relevant DoD needs

The first phase of the TASK program (2000-2002) focused on the demonstration of initial multi-agent system concepts through the application of a particular technology (game dynamics, control theory, genetic programming, information theory, etc.) to one or more critical elements of a Multi-Agent Systems (coordination, adaptation, agent construction, etc.) within the context of one of the three REFs. The second phase of the TASK program (2003-2004) is focused on a proof-of-concept application of new algorithms and technologies within a single coherent Open Experimentation Framework consisting of a set of highly DoD relevant UAV surveillance problems and a multi-agent system design, evaluation, and experimentation methodology. A third phase of TASK (2004+) might conceivably take the results of the second phase and apply them to transition-oriented DARPA programs (similar to MICA, FCS and Sense-It) and/or to a new program involving the autonomous control and coordination of unmanned vehicles in highly dynamic environments.

3 OEF

The TASK OEF serves as a common problem framework within which multiple researchers can evaluate different approaches to MAS design and analysis. Of the seven defined critical elements, Adaptation, Control, Coordination, Uncertainty, Resource Management, Agent Construction, and Reliability, our initial emphasis is on Adaptation and Coordination. OEF efforts have been focused on:

- Defining MAS design problems, emphasizing Adaptation and Coordination
 - “Baseline problems” - the simplest problem that captures the essential characteristics of the critical element

- Extended problems have parameters to address robustness, scalability, mission complexity, uncertainty
- Defining OEF/MAS Challenge Experiments
 - “Baseline experiments” for quantitative evaluation of design approaches
 - Individual experiments are defined/designed by the individual projects
- Identify and develop metrics to evaluate MAS solutions to the OEF design problems

Adaptation:

We define MAS adaptation as the ability of the MAS to recognize and respond to unexpected change. New theories must be developed for adaptive distributed control as well as new design and evaluation methodologies. Without adaptation agents are unable to react effectively to unanticipated change and the performance of a MAS may degrade unpredictably. Within the context of the OEF problem, one would expect to see a degraded ability of the UAVs to service tasks and complete missions as change is introduced, yielding unpredictable mission behavior and/or performance. Solutions may range from parametric adjustment to evolutionary changes in the structure of the system (learning). The rate and character of adaptation solutions needed may differ with the rate and character of change, such as dynamics associated with threats and tasks, failures and noise, and adversary behavior. Our goal for the OEF has been to develop one or more design problems (e.g. MAS adaptation problems) that

- 1) Capture the “essence” of the adaptation problem and, as such, are useful to algorithm/agent/MAS designers
- 2) Provide a way to categorize the class of algorithms that can solve the problem
- 3) Are as simple as possible, yet lead to non-trivial MAS interactions and behavior
- 4) Tie to the OEF (UAV swarms) and other critical DoD MAS systems
- 5) Support experiments to evaluate algorithm/technique effectiveness

Coordination:

We define MAS coordination as the propagation of information and local agent action and reaction to propagated information and remote agent actions. Without coordination, agents unintentionally waste their efforts and squander resources or fail to accomplish objectives that require collective effort. Within the OEF context, agents might unnecessarily duplicate servicing of tasks, tasks could go unserved, service may not be timely, or resources could be used inefficiently.

Since the set of MAS coordination strategies may change as environmental uncertainty or bandwidth changes, the research goals are to identify: the salient design space attributes, the

dimensions in which coordination needs to scale, and metrics for MAS performance with and without alternate coordination mechanisms.

3.1 OEF History

3.1.1 IVRS

ALPHATECH began the TASK program by creating and managing the Intelligent Vehicle and Roadway System (IVRS) Research Exploration Framework (REF). IVRS was a problem designed to entail multiple vehicles controlled by autonomous agents interacting in an environment.

The essential components of the IVRS were a population of vehicles and a roadway system on which the vehicles move. Another component of primary importance, but not essential in all situations of interest, is a distributed population of “traffic signals” with some capability to regulate the flow of vehicles. Thus, the IVRS represented an example of a system characterized by the flow of particles through a conduit system, possibly regulated by valves. Systems such as these are capable of exhibiting microscopic and macroscopic behaviors of interest including coalition formation, agent-environment interaction, wave phenomena, and other flow patterns. Moreover, within the IVRS setting we had the freedom to assign a rich set of behaviors, adaptation laws, and interaction mechanisms to the individual particles and valves (agents).

The IVRS problem supported agent research across the following dimensions:

- **Autonomy**
 - Decisions made by individual vehicles
- **Interaction & Communication**
 - Message passing (nearby agents)
 - Negotiation/protocols
 - Chalkboard
 - Information dissemination
- **Vehicle Adaptation & Dynamics**
 - Obstacle and collision avoidance
 - Learning & memory
 - Route planning
 - Reaction & dynamic replanning
 - Attractive & repulsive forces
 - Social rules & fairness
 - Feedback mechanisms
- **Vehicle Goals (self-interest)**

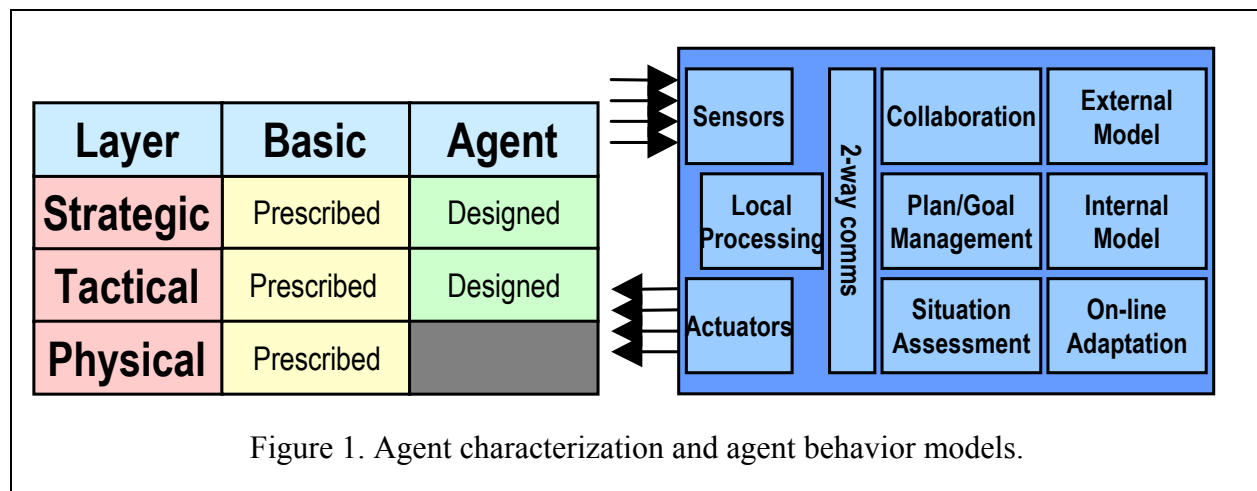
- Reach destination
- Minimize elapsed travel time
- Maximize safety
- Maximize travel comfort (relatively constant speed & direction)
- Minimize fuel usage, other costs
- Vehicle Equipment & Performance Specs
 - Size (mass, length, width, height)
 - Speed / acceleration / deceleration
 - Fuel economy & emissions
 - Sensors
 - Communications
- Heterogeneous Vehicles
 - Varying priorities/values for goals
 - Varying equipment & performance specs

The IVRS REF also supported the following MAS research areas:

- Collective and Global Goals
 - Maximize aggregate throughput
 - Maximize aggregate safety
 - Minimize aggregate pollution
 - Minimize aggregate fuel usage
- Dynamic Environment
 - Each individual vehicle encounters a rapidly changing environment
 - other vehicles
 - transmitted messages
 - intersections, curves, & grades
 - obstacles & hazards
 - equipment failures
 - control agents
 - rules/laws/protocols
 - evolving goals
 - evolving aggregate conditions
- Possible Control Agent Features
 - Monitoring and dissemination of aggregate data
 - Self-motivated to achieve localized collective goals
 - control via behavioral mechanisms
 - control via supervisory signals
- Emergent Behavior

- Bifurcations, chaos, & phase changes
- Self-organizing characteristics
 - coalition formation (e.g., platoons)
 - fast & slow lanes
- Flow characteristics & patterns
 - laminar vs. turbulent
 - steady vs. unsteady
 - “slinky effect”

Within the IVRS REF six research groups conducted a wide variety of critical multi-agent system research including: investigation of decentralized agent control techniques that guarantee emergent behavior (Dartmouth, “Exit at RFK stadium” problem); construction of traffic signal agents from Elementary Adaptive Modules in order to improve vehicle flow (MIT/BBN); investigation of platooning behaviors in non-stationary environments (USC/ISI), investigation of the effect of different reinforcement learning-based control techniques on emergent behavior (ALPHATECH), the evolution of transport network agents (Hampshire College) and many other results.



As part of the establishment of the IVRS REF we developed a framework to characterize agent subsystems and behavior characterization. The agent behavior characterization recognized a need to separate the range of agent behaviors into three layers: physical, tactical, and strategic. Physical level behaviors govern low-level dynamics of the agent (e.g. motion control, collision avoidance) and are “autonomic” in nature. Strategic level behaviors govern “long-term” goal-oriented planning for the agent. Tactical level behaviors govern short-term, or tactical, decision-making to achieve immediate objectives and are the primary focal point of multi-agent system research.

The agent design subsystems included in the framework are:

- Collaboration: governs cooperative and competitive mechanisms toward achieving individual and common goals.
- Plan and Goal Management: governs dynamic management of goal priorities and plans to achieve local and common goals.
- Situation Assessment: governs local observation and instantaneous assessment of individual, group, and environmental state.
- External Model: governs formulation and dynamic updating of external world model including prediction of external behavior and impact of local decisions.
- Internal Model: governs autonomous actions to achieve individual performance objectives.
- On-line Adaptation: governs dynamic modification of properties and behaviors within pre-defined constraints in response to assessed situation.

The essential properties and default behaviors of the physical agent characterization serve as a foundation for constructing the agent characterization. Inputs to the agent subsystems must be compatible with prescribed low-level sensor capabilities; outputs from the agent subsystems must be compatible with prescribed actuator capabilities; collaboration mechanisms must be compatible with prescribed communication capabilities.

3.1.2 CAHDE

The IVRS REF was later transitioned into the Control and Adaptation of Heterogeneous Agents in Dynamic Environments (CAHDE) REF, which kept many of the interesting aspects of the IVRS problem, but directed research to a 3D multi-UAV problem, which had more military relevance.

The CAHDE REF is designed to support research in which vehicle agents compete and/or collaborate to minimize undesirable emergent behavior, optimize the use of a shared resource, and attain their individual and/or collective goals. The CAHDE REF problem is M agents servicing N tasks, such as a team of UAVs gathering intelligence, teams of smart munitions taking out targets, or the coordinated airlift of supplies. Research goals included enabling vehicles to adapt to dynamic environments, such as weather and threats, and subsequent flexibility and adaptation of the MAS, through coordinated action and interaction.

As part of the establishment of the CAHDE REF we refined the agent/MAS characterization developed during the IVRS REF into the beginnings of a more formal design and analysis framework. Collectively, the CAHDE REF projects started to enumerate the critical design and evaluation dimensions of MAS, namely propagation of information, adaptation, coordination, control, uncertainty, performance guarantees, and local versus global objectives. (This would later develop into the critical MAS dimensions of the OEF: adaptation, coordination, and autonomy, plus agent construction). The CADHE REF also began producing a set of comprehensive metrics for the design and analysis of MAS along with some early prototype tools.

As with the IVRS REF the seven research groups participating in the CAHDE REF were able to, separately and collaboratively, conduct a wide variety of critical multi-agent system research including:

- Improved responsiveness of adaptive MAS relative to non-adaptive MAS through the recognition of unanticipated situations and response by evolving emergent behaviors. [MIT/BBN and Hampshire College]
- Methods for reasoning about large-scale agent systems [UIUC]
- Library of Elementary Adaptive Modules and tested structures [MIT/BBN]
- PushPop Evolution components [Hampshire College]
- Metrics for coordination and collaboration as generalized forms of agent-agent synchronization [SFI]
- Market-based and game-dynamical approaches for decentralized control of large-scale MAS in dynamic environments [USC/ISI]
- Reinforcement learning and stochastic control agent design [ALPHATECH]
- Distributed algorithms (using artificial potential functions) for 3D airspace control enabling agents to adapt to their neighbors (and MAS to adapt to their environment) [Dartmouth]
- And many others

ALPHATECH's success with the establishment of the CAHDE REF and the integration of the research of a subset of the program's PIs was the impetus behind the current OEF, which is an extension of the CAHDE REF problem. The following sections describe the OEF design problems in detail.

3.2 Design Problems

Within MAS designs, two major issues always faced by MAS designers are the selection of coordination and adaptation mechanisms appropriate to the characteristics of the problem at hand. The long history of computer science research in MAS coordination chronicles many theories, methodologies, and implementations from which to choose but provides only loose guidelines for selecting one approach over another. Generally, coordination between autonomous agents within a MAS is used to complete in some optimal manner a task that is beyond the capabilities of an individual agent. As such, coordination is closely associated with team design and spontaneous team formation, many variations of plan creation and execution, situation monitoring and active re-planning, coordinated behavior responses (in MASs that do not include an explicit "planning and re-planning" mechanism), and joint-coordinated learning. Adaptation has nearly as long a history of research but with fewer theories, etc. from which to choose. MAS adaptation must handle changes to the environment in which the MAS operates, changes to the MAS itself (such as changes to the number and capabilities of the autonomous agents), and new MAS-wide tasking. Of course, in MAS design adaptation and coordination are closely coupled.

Currently, the TASK OEF Adaptation and Coordination MAS Design Problems parameterizes the problem space with respect to *uncertainty*, *scale*, *mission complexity*, *responsiveness*, and *reliability* and the solution (design) space with respect to *adaptation*, *coordination*, and *autonomy*¹. These parameterizations represent many of the situations in which MAS solutions are applicable: in particular, many military operations that might employ agents can be parameterized in this manner. The Design Problems and corresponding Experiments are specified with respect to *baseline* values for the problem space parameters. Measurements of domain dependent quantities, which correspond to the important domain dependent problem space *goals*, are carried out in standard testbed simulations. When experiments are conducted that systematically vary the baseline problem space parameters, the measurements provide quantified responses of particular Coordination and Adaptation algorithms to different problem situations.

A major difficulty one faces when specifying experimental MAS Design Problems is to avoid dominance of the solution space by the details and intricacies of specific details of the domain in which the problems are posed while retaining the features essential to the MAS characterization of the domain. Of course, different problem domains, no matter how abstractly stated, will always emphasize some problem and solution space parameters over others and are likely to include parameters not noted above. The first step in our approach to design problem and experiment specification is *top-down* in which we state as abstractly as possible a general MAS design problem that isolates as much as possible the essential features of Coordination and Adaptation. The second step is *bottom-up* in which we select a problem domain for experiments, which introduces domain dependent instances of the problem and solution space parameters.

3.2.1 The General TASK OEF Design Problem and Experiment

The abstract MAS Coordination and Adaptation Design Problem is m autonomous agents, which can exchange information, detect tasks, and provide some type of *service*² to a detected task, that find and service n tasks. For the *baseline* design problem and experiments we specify the following additional constraints:

1. Each task is serviced only once by only one agent
2. Each task has unique *identification* and *location*³
3. An agent may know the *location* of a task
4. An agent may or may not know the locations of *all* tasks

¹ For an approach to the measurement of *agent autonomy* in Multi-Agent Systems see, K. S. Barber and C. E. Martin, *Agent Autonomy: Specification, Measurement, and Dynamic Adjustment*, in *Proceedings of the Autonomy Control Software Workshop at Autonomous Agents 1999 (Agents '99)*, (Association for Computing Machinery), pp5-15, 1999.

² *Service*, *distance*, and other such terms are intentionally ambiguous: *service* ranges from only detection through determination of the properties of a detected task, doing *something* to the task, to complete *destruction* of a detected task, which, of course, is also ambiguous.

³ The meanings of *identity*, *location*, and *distance* are domain dependent.

5. The mixture of *known* and *unknown* tasks ranges from *all known* to *all unknown*
6. During an experiment the mixture of know and unknown tasks may remain constant or it may vary slowly, disruptively (suddenly), or randomly. Note that a Coordination and Adaptation approach must encompass mixtures of known and unknown tasks, that is, it should not deal with only all known or all unknown.
7. An agent learns the *identification* of a task upon *detection* of the task
8. A task can have either one x_1 or two *type attributes* x_1x_2 , where $x_1, x_2 \in \{A, B, C, D, F\}$
9. Detectors are of a *single type* y , where $y \in \{A, B, C, D, F\}$, and can detect only those tasks that have the same type attribute
10. An agent can have two detectors each of a different type
11. Tasks have *values*, which an agent can *earn* for task detection-service
12. Either no communication between agents or only binary agent-to-agent communication, which is limited to a standard *distance*² (communication range) between the two agents.
13. In some experiments, there are range-limited *broadcasts* by an agent (initially there are no broadcasts)

Variations from the baseline problem are the subject of later discussions.

The baseline experimental measurements or solution measures are divided into two groups.

Solution Measures: Mission Success

1. The rate at which an agent services tasks
2. The rate at which an agent services previously serviced tasks (serviced by other agents)
3. The length of time before a task is serviced

Solution Measures: MAS Performance

1. The *rate*⁴ at which agents communicate
2. The average *complexity*⁵ of inter-agent communications
3. The average *length*⁶ of inter-agent communications
4. The rate at which an agent consumes *computational resources*⁷

Of course, in general additional measurements may be made and are likely to be required in a particular domain.

In addition to the constraints noted above, additional problem space parameters and ranges, which a treatment of the OEF baseline Coordination and Adaptation problem should handle and

⁴ The *rate* of inter-agent communication is taken to be the number occurrences per unit time

⁵ The *complexity* of inter-agent communications is TBD.

⁶ The *length* of an inter-agent communication is taken to be the number of four-bit bytes.

⁷ The definition of *computational resources* is TBD.

which will be varied from experiment to experiment as well as during experiments, are the following:

1. The number of tasks and the number of agents
2. *Failure*, which is the disappearance of one or more agents

The purpose of the TASK OEF baseline design problem as described above is to provide a *defined, well-known* standard environment in which the rates achieved by different coordination and adaptation approaches can be measured. For a particular coordination and adaptation mechanism, standard systematic variations from the baseline problem parameters (as discussed above) provide the measures of the specified rates under different problem conditions. MAS designers can use the experimentally derived mappings between the problem space parameters and the measured rates to assess the potential performance of specific coordination and adaptation mechanisms in particular problem environments.

Variations from the **baseline** Coordination and Adaptation Design problem include the following:

1. A task of two types, x_1x_2 , must be detected simultaneously by an agent of type x_1 and an agent of type x_2
2. Agent-to-agent communications are *unreliable*
3. The information communicated by an agent may be *inaccurate*
4. An agent's detection-service of a task may be inaccurate with respect to location, identity, and/or type
5. A task may change its location unannounced to the MAS. In this case, the agents can detect and track task movement
6. A task can detect and track agents and can change its location based upon to this knowledge
7. A task may need to be detected-serviced on a schedule
8. A task that can detect and track can be a *threat*, that is, such a task can remove an agent from the MAS
9. An added task with an unknown location (identity, and type) must be found within a specified time interval after its addition

These variations, which require domain-specific details, are the basis for addition Coordination and Adaptation design problems in the Unmanned Aerial Vehicle (UAV) domain.

3.2.2 The UAV Domain UAV-S (1) Baseline Problem and Experiment

Within the military context of the TASK OEF the simplest realization of the general baseline Coordination and Adaptation Design Problem is a MAS of m autonomous-agent-driven UAVs and n ground sites. The UAVs are outfitted with sensors and communication gear and the MAS

is *ordered*⁸ to detect and service the ground sites. As discussed more fully below, in the UAV domain we assume that detectors have an additional attribute (different from the *type attribute y*) that can have the value of either *surveillance* or *search*. A *surveillance detector* determines the *identity*, *type*, and precise *geo-location* of a ground site and has a small field-of-view (FOV). A *search detector* determines only that a ground site exists within its FOV, which is significantly larger than a surveillance detector's. A *search detector* performs *detection*, and a *surveillance detector* performs *service*.

As discussed above the 0th variation of the baseline UAV problem has the following conditions:

1. A constant specified mixture of ground sites with geographic locations known by every UAV and of ground sites with geo-locations unknown by any UAV
2. The number of UAVs and the number of ground sites remain constant
3. A serviced ground site, after a specified standard interval of time, is replaced by a new ground site, which has a new and unique identity and location^{9,10}
4. There is to be no duplicate surveillance (servicing) of any site by the same or any other UAV
5. A site can be found (by the search detector) multiple times by any UAV
6. There are multiple types of UAVs, that is, the MAS is composed of UAVs that have different flight and performance characteristics¹¹
7. Each detector and each ground site has a single type, which is one of $\{A, B, C, D, F\}$
8. Each ground site has a *decimal value* in the range $[0 \dots 100]$
9. A UAV is aware of all other UAVs that are within a specified distance and is not aware of those outside this distance
10. A UAV can communicate with only one other UAV at a time
11. Two UAVs can communicate only when they are closer than a specified distance

The experimental measurements are the following:

1. The average rate at which the UAVs service (through the surveillance detector) sites
2. The average rate at which the UAVs service previously detected sites

⁸ *Ordered* in the military sense of “You guys go do this now!”

⁹ The location of a new ground site, which replaces a detected ground site, may be made known to all the UAVs. A serviced, aged site is removed from the simulation upon replacement in order to keep the landscape from becoming littered with inactive sites.

¹⁰ The continuous replacement of detected, aged sites has three purposes: (1) to avoid experiments dominated by the initial configuration of the ground sites—the *cold start* effect, (2) to examine problems that can not be treated, perhaps more easily, by centralized optimization computations, and (3) to replicate the quasi-real-world environment where it makes sense to deploy MASs.

¹¹ Purely homogeneous MASs (all UAVs have the same flight and performance characteristics) are included in the systematic variation of the problem parameters.

3. The time elapsed before a site is first serviced
4. The average rate at which the UAVs detect (through the search detector) sites
5. The average rate at which UAVs communicate
6. The average length of the UAVs' transmissions
7. [Future: the average complexity of communications]
8. [Future: the average computational resources consumed per UAV]

This fairly simply problem supports a rich array of research into multi-agent systems –

- From handling uncertainty in the environment – e.g. resulting from an agent's limited communication and sensor ranges
- To the investigation of coordination strategies under a range of different scenarios – such as highly bandwidth constrained scenarios and scenarios involving swarms of agents
- From the investigation of different control strategies – such as leader-follower, hierarchical control, autonomous control, market-based controls, and so on.
- To the investigation of strategies to adapt agent/MAS behaviors to: changes in numbers of agents, changes in task mix, changes in agent heterogeneity/specialization, etc., and
- From investigating MAS robustness to investigating MAS scalability.

Additional physical characteristics and constraints are required in order to complete a fully specified experiment that can be reproduced in different implementations of testbeds and simulators.

1. The physical space for OEF experiments, known as the Area of Interest (AOI), is a ground surface rectangle 400km per side and unlimited altitude. There are no terrain features such as mountains, roads, rivers, lakes, etc.
2. A site is a *point* on and within the AOI ground surface rectangle. A standard coordinate system¹² will not be specified since the geophysical measurements enter in only *relative* or *difference* terms. The *distribution* of sites within the AOI or at a minimum the *characteristics* of the distribution must be repeatable for the experimental measurements made with different MAS design approaches to be comparable. For OEF surveillance and search experiments, the site locations must be distributed *randomly, continuously, and uniformly* across the AOI rectangle¹³.

¹² One can find definitions for the standard geographic reference systems used by DoD on the National Imagery and Mapping Agency (NIMA) WWW site <http://www.nima.mil>. Of particular interest is the Digital Terrain Elevation Data (DTED standard (MIL-PRF-89020b), which uses the standard World Geodetic System (WGS 84) (MIL-STD-2401) for horizontal datum and Mean Sea Level (MSL) determined by the 1996 Earth Gravitational Model.

¹³ With a random continuous uniform distribution we avoid the detailed specification of configurations of site locations and simplify the generation of new locations after an aged detected site is removed.

$$P(x, y)dxdy = \frac{1}{x_{max}} \frac{1}{y_{max}} dxdy \quad \begin{cases} 0 \leq x \leq x_{max}, 0 \leq y \leq y_{max} \\ \{x < 0, x > x_{max}, y < 0, y > y_{max}\} \end{cases}$$

where $P(x, y)$ is the continuous differential probability that a site is located within the differential $dxdy$ of the point (x, y) .

3. The mixture of known and unknown sites is 80% sites whose geo-location is known to all UAVs and 20% whose geo-location is unknown by all UAVs.
4. The target *type* (T) of a site is randomly and uniformly selected to be one of (A, B, C, D, F) .
5. The *value* of a site is a real number randomly and uniformly selected from the interval $(0 \dots 100)$.
6. A ground site vanishes 60 seconds after detection.
7. There are always a total of 100 ground sites (detected and undetected) in the AOI.

While we have attempted to remove the major physical details of the high-fidelity UAV search and surveillance problem, there are some details of both sensor and air vehicle types that must be examined and included.

The initial *surveillance* task of the problem—precise detection of known ground sites—and the later *search* task of the problem present a slight conundrum. Current Synthetic Aperture Radar (SAR) in its *Spotlight* or *Spot* mode is often used for high accuracy resolution (0.1m to 1.0m) detection¹⁴ from UAVs. The size of the ground patch detected by the Spot SAR is roughly 500m to 800m in diameter and generally 4km to 25km away on one side of the air vehicle. UAV SAR in its *Stripmap* mode, which might be used for search, detects a swath ~1km wide (with resolution 0.3m to 3.0m) 7km to 30km to the side of the UAV. *Ground Moving Target Indication* (GMTI) radar, which is different from the SAR, cuts a swath ~10km wide 4km to 25km to the side of the air vehicle and has a minimum target speed detection of ~3m/s¹⁵.

8. As a compromise between retaining essential features and excluding extraneous details, we take the detection footprint size for the *surveillance detector* to be a circle with a 300m diameter directly beneath the UAV, that is, centered on the UAV's nadir vector. The *surveillance detector* detects the unique *identity*, the precise *geo-location*, and the *type* of a ground site.
9. If a surveillance detector is of a different *type* than the ground site, the site remains unserved, and the UAV does not accrue the value of the site.
10. A site must remain within the surveillance detector's footprint or FOV for 10sec to be serviced

¹⁴ *Detection* is both accurate geolocation determination and target type identification: is it a tank, a rocket launcher, a Starbucks, etc.

¹⁵ Other SAR modes such as Stereoscopic SAR or Digital Elevation Model (DEM) construction and for feature (building, water tank, road, etc) extraction and Interferometric SAR for terrain change detection are also useful but will not be considered here. Similarly, optical, infrared, video, and Signal Intelligence (SIGINT) sensors will be excluded.

11. The instantaneous *search detector* for **static** sites is a circle 1000m in diameter also centered on the air vehicle's nadir. The *search detector* can detect only the presence of a site within its search circle and determine the site's geo-location to within 6m: it **cannot** detect the *identity*, precise *geo-location*, or *type* of the site.
12. The search and surveillance detectors **cannot** operate simultaneously: the switch from one detection mode to the other consumes one-half second (0.5sec).

The GMTI specifications are discussed in the Design Problems for Moving Targets section.

Several *types* of UAVs can be differentiated by *flight characteristics*. The details of these different types may influence significantly the performance of various MAS Coordination and Adaptation approaches. First, there are fixed-winged UAVs, that can not hover, can not ascend or descend without moving forward at significant speed, and can not turn sharply, and there are non-fixed-winged UAVs that can hover, can ascend or descend without moving forward, and can turn sharply. Second, some UAVs are designed to perform long-endurance general missions, and others are designed to perform short-duration tactical special-purpose missions. Third, some UAVs operate at high to medium altitude and others at low to medium altitude. Fourth, some UAVs are large and accommodate multiple sensors, complex navigation devices, and sophisticated communication packages, while others are small to micro and accommodate one or two limited capability sensors, rudimentary (but perhaps quite innovative¹⁶) navigation, and limited communications. Fifth, some UAVs are fast and others are slow.

Actual and proposed UAVs encompass nearly all combinations of these characteristics. For example, the Global Hawk UAV has a fixed wing, operates at high altitudes (~20km) on long-duration (~35hours with a range of 22,000km) endurance missions, is relatively fast (~180m/s), and is "large" (35m wingspan and 13m length). The Predator UAV also has a fixed wing, operates at moderate altitudes (~8km) on short (range ~740km) missions, is slow (cruise speed ~38m/s), and is roughly the same size as the Global Hawk. Of course, the Predator can be armed. By contrast, the Dragon Eye UAV (used by the Marine Corps for over-the-hill reconnaissance) can be carried in a backpack (weighs ~2kg) and assembled in the field, (possibly) operates up to 0.1524km altitude for up to 1 hour at speeds up to 18m/s, and has a wingspan of 1.143m. It navigates by GPS waypoints (field-in-flight programmed), carries full-motion color, low-light, and infrared video cameras (not simultaneously), and can transmit line-of-sight video up to 10km. Current examples of small rotary-winded UAVs are Schiebel Camcopter (100km range, 6 hour endurance, 25kg payload) and the Yamaha RMAX (10km range, 90 min flight time, 30kg payload). More exploratory UAVs include DARPA's Organic Air Vehicles (OAV), which are small—up to 28 inches diameter—duct-fan VTOL hovering craft, short-duration, and potential tactical sensor platforms, and Hummingbird Warrior, which is a medium altitude, long-duration endurance, moderately fast VTOL with a 3700km range.

¹⁶ For example see *An Ultra Wideband Radar for Micro Air Vehicle Applications* by Robert J. Fontana, et al, available at http://www.multispectral.com/pdf/Advances_Radar.pdf.

For MAS Coordination and Adaptation design and experimentation, we define only two types of UAVs with respect to flight characteristics: a fixed-wing (UAV Alpha) that corresponds roughly to the Predator and a VTOL hoverer (UAV Beta) that corresponds roughly to the Fire Scout. Both of these will operate at medium altitudes and have medium- to long- duration.

Name: UAV Alpha¹⁷

Description: Fixed-winged

Minimum Speed:

Maximum Speed: 60 m/s (level flight)

Cruise Speed: 38 m/s (level flight, dwell speed)

Maximum Forward Acceleration:

Maximum Climb Rate:

Maximum Descent Rate:

Minimum Turn Radius: 30 m at 38 m/s 82° bank, 1000 m at 38 m/s 30° bank,
300 m at 60 m/s 82° bank, 1300 m at 60m/s 30° bank

Maximum Altitude: 7.6 km

Fuel Capacity: 300 kg

Fuel Consumption:

Empty Mass: 715 kg

Fueled Mass: 1,015 kg

Width: 15.0 m (wingspan)

Height: 2.0 m

Length: 8.0 m

Name: UAV Beta¹⁸

Description: Non-fixed-winged, VTOL, hover

Minimum Speed:

Maximum Speed: 64 m/s (level flight)

Cruise Speed:

Maximum Forward Acceleration:

Maximum Climb Rate:

Maximum Descent Rate:

Maximum Altitude: 6 km

Fuel Capacity:

Fuel Consumption:

Empty Mass: 72 kg

Fueled Mass:

¹⁷ A detailed mathematically specified flight kinematics model for the actual flight dynamics of the fixed-wing UAV Alpha will be provided.

¹⁸ A detailed mathematically specified flight kinematics model for the actual flight dynamics for the non-fixed wing, VTOL, hover UAV Beta will be provided.

Width: 8.4 m (rotor)
Height: 2.9 m
Length:

- There are, for example, 16 UAVs: 8 of type UAV Alpha and 8 of type UAV Beta
- The maximum UAV-to-UAV communication distance is 3.2 km
- A UAV can determine the presence, heading, and speed of all other UAVs within 6km

The following four problems are natural extensions to the baseline UAV-S(1) problem. Each problem is designed to stress the MAS solutions in a new way and will be the subject of coordinated PI research efforts as the research evaluation progresses.

1. UAV-S Problem Class - UAV-S (2) Problem: Cross-Mission Tasking

- *UAV-S (2) Problem complexity:* Same problem as UAV-S (2), with the following change:
- Tasks are of type x or xx, $x \in \{A \dots F\}$ (e.g. A, CD, E, AB). XX Tasks must be serviced simultaneously (within time window $[t_0, t_1]$) by different agents with appropriate capability - e.g. CD task is serviced by AC agent and D agent, or CE agent and AD agent, but not by a single CD agent.
- *UAV-S (2) MAS Objective and Problem Solution Evaluation:* Same as UAV-S (1)

2. UAV-S Problem Class - UAV-S (3) Problem: Imperfect Information

- *UAV-S (3) Problem complexity:* Same problem as UAV-S (2), with the following change:
- *Detection and Identification are imperfect.* [TBD - parameter specification. Some agents will be higher fidelity than other agents].
- *UAV-S (3) MAS Objective and Problem Solution Evaluation:* Same as UAV-S (2)

3. UAV-S Problem Class - UAV-S (4) Problem: Mobile Targets

- *UAV-S (4) Problem complexity:*
 - Fixed number of agents (m) of one type, Fixed number of tasks (n) of one type. Tasks are uniform in value.
 - Each task needs to be visited repeatedly (e.g. to maintain location estimate).
 - Detection and identification are accurate. Each task has a unique ID.
 - Initial position of task is known, but are subsequently unknown due to mobility.
 - Task mobility characteristics may be known (by the agents) or unknown, fixed or variable (within a task: e.g. go-stop-go), heterogeneous or homogeneous (across the tasks).
- *UAV-S (4) MAS Objective:* Maintain position estimate on each target
- *UAV-S (4) Problem Solution Evaluation:* Measure the resources required by the UAV-S design problem solutions

4. UAV-S Problem Class - UAV-S (5) Problem: Hybrid Tasking

- Definition TBD

3.3 Milestones Achieved and Future Direction

ALPHATECH's work in defining and managing the OEF has been successful, measured by a participatory process that resulted in the definition of a set of baseline problems with detailed parameters, within which the TASK PIs can cast their work. ALPHATECH has taken substantial steps to engage PIs in the specification of the OEF, which has resulted in a specification that is both relevant to military problems and capable of hosting a variety of research approaches.

Our future OEF work will be oriented around supporting the PI researchers through the following:

- Developing a parameterized problem generator that will allow the PIs to specify parameters through a set of menus and will produce a file with a specific problem configuration consistent with the UAV-S (1) surveillance problem
- Design baseline approaches to baseline problems for research evaluation purposes
- Further definition of the more advanced UAV-S problems
- Continued coordination of PI research within the OEF context

4 Testbed for Taskable Agent Systems

4.1 Overview

The Testbed for Taskable Agent Systems (TTAS) is a simulation platform with which we can perform experiments on different designs and implementations of Multi-Agent Systems (MASs). TTAS supports our own research work by providing a mechanism for evaluating disparate agent designs, operating together as a Multi-Agent System. The inter-agent communication capabilities provided by TTAS allow us to conduct research in MAS coordination and the multitude of configurable parameters supports our work on agent adaptation. Additionally, TTAS provides facilities for capturing and storing detailed experiment data and supports analysis through repeatable experimentation. TTAS is available to all TASK PI researchers as an environment for evaluating disparate approaches to agent technology within a common MAS problem framework. TTAS can be used by all project PIs to test and analyze their particular approach to coordination or adaptation and then the results can be compared across the TASK program as a whole. This strategy will provide enormous benefit in terms of the ability to classify the applicability of various technologies across the dimensions of an entire class of problems.

Our focus with TTAS is on coordination and adaptation in a MAS of autonomous Unmanned Aerial Vehicles (UAVs) that search for and survey known, unknown, stationary, and mobile ground targets (tasks) and that can communicate. TTAS has a set of well-defined Application Programming Interfaces (APIs) for the UAVs, agents, tasks, agent-to-agent messaging, and data

recording. Any MAS that implements these APIs can execute within TTAS. Agents from different approaches can execute simultaneously within TTAS and, if they adhere to TTAS' basic messaging, can function as a MAS composed of heterogeneous agent designs.

TTAS is fully three dimensional with six full degrees of freedom for UAV motion (three translational and three rotational about the UAV's body center) and three-dimensional terrain location for the ground sites. TTAS uses flight simple kinematics models for UAV motion rather than full equation of motion flight dynamics models, which is sufficient for OEF experiments.

TTAS uses an XML-based configuration file for data input and to record the conditions of a repeatable OEF Experiment. The experimenter records the parameters of a simulation experiment within the configuration file, which TTAS reads, interprets, and executes. These parameters describe the following:

1. The simulation environment
2. The UAVs
3. The agents that control the UAVs
4. The flight kinematics models of the UAVs
5. The sensor constraints of the UAVs
6. The target ground sites (tasks)
7. The agents that controls the ground sites (if any)

The configuration file and how TTAS uses it is described in Section 3.2.

If enabled by the experimenter, data is recorded into a relational database. At execution time the experimenter can select the specific data to be recorded and the frequency of recording but may need to have implemented methods within specific Java classes (the *agent*, *UAV*, *ground site*, etc) to extract the data encapsulated within the classes. These extraction methods are defined as part of the TTAS APIs.

The progress of an experiment can be monitored in a two dimensional *wizard's-eye-view* that displays a downward look into the AOI with identification annotating the UAV and ground target symbols. Also, a fully three dimensional view with user maneuverable viewpoint is available but currently is of limited utility.

The quantities varied during normal OEF experiments are the following:

1. The number of UAVs
2. The number of target ground sites
3. The ratio of the number of ground sites about which the UAVs know (known tasks) to the number about which they do not know (unknown tasks)
4. The maximum distance at which one UAV can detect another UAV

5. The maximum distance at which two UAVs can communicate
6. Errors in UAV to UAV communication
7. The size of a sensor footprint
8. The distance at which a sensor can detect a target ground site
9. The accuracy of a sensor
10. The reliability of a sensor

Other quantities such as fuel capacity and communication bandwidth will be added.

Summary quantities measured during normal OEF experiments are the following:

1. The rate at which known ground sites are surveyed once
2. The rate at which known ground sites are surveyed more than once
3. The rate at which unknown ground sites are found
4. The average time interval between the appearance of a known ground site and is surveillance
5. The average time interval between the appearance of an unknown ground site and is discovery

Of course, additional quantities can be computed from the recorded data. Experimenter implemented summary quantities will be added.

4.2 Technical Description

TTAS is built upon the *Multi-Agent Development Kit*¹⁹ (MadKit), designed and implemented by Jacques Ferber, Olivier Gutknecht, Fabien Michel at Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (*LIRMM*). MadKit provides all the basic agent services that TTAS requires: agent identification, lifecycle, and messaging. We surveyed many platforms for multi-agent simulations and chose MadKit because (1) it is written in Java, (2) it imposes no significant restriction on how an agent is designed and implemented, (3) it can be extended easily, and (4) it has provisions for data gathering. We discuss the aspects of MadKit relevant to TTAS in Section 3.2.1 and TTAS's use of MadKit throughout Section 3.2.

TTAS's design is object-oriented and highly modular. TTAS' overall design and the design of its components is performed and maintained in the *Unified Modeling Language* (UML). Complex individual component logic is designed and archived with standard *flow charts*. *Unit testing* is used extensively during implementation.

We achieve TTAS's modularity by exploiting Java's dynamic class loading and reflection capabilities. Many of the parameters in the configuration file consist of the complete name of a

¹⁹ MadKit is open source software, available under the standard LGPL and GPL licenses, and can be obtained from <http://www.madkit.org>.

Java class, such as `com.alphatech.TASK.uav.UAVBetaController` (in the case of the rotary winged hoverable air vehicle) and the parameters necessary to create instances of this class, such as `com.alphatech.TASK.basicvehicle.BasicVehicleState`, `com.alphatech.TASK.uav.UAVBetaParameters`, etc. Other class parameters are simple numbers, such as the size of the simulation world's Area of Interest (AOI) or the coordinates of the initial position, velocity, and direction of a UAV. The Java Virtual Machine (JVM) loads these Java classes into TTAS at runtime. With runtime class loading the same UAV can be controlled by agent implementation A in one experimental run and by agent implementation B in another without any code changes to TTAS or any changes in the experiment's configuration. TTAS uses Java *reflection* to retrieve the *constructors* and *methods* of a particular class appropriate to the types of the particular parameters supplied in the configuration file. Reflection is necessary since Java constructors and methods can be overloaded. New *instances* of a class parameterized with respect to the values of the specified parameters can be created. Calls on the methods retrieved by reflection can be used to set additional parameters. Also, TTAS uses Java *cloning* to create new instances of a class as a *deep copy* of an existing instance. As will be discussed below, this method is used extensively to generate new UAVs or target ground sites on an experimenter-specified schedule.

TTAS' high level design and major components is depicted schematically in Fig. 2.

TTAS's components are discussed below.

4.2.1 Multi-Agent Development Kit

MadKit is implemented as a *micro-kernel* with the services noted above implemented as MadKit *agents* that work directly with the micro-kernel. The MadKit micro-kernel is small (50 kilobytes) and can be executed on many Java different platforms (J2SE or J2EE 1.4 and Personal Java, for example). A developer can extend agent services by adding special agents that work with the micro-kernel directly.

A MadKit based MAS implementation can be executed on a single platform or on multiple platforms that communicate by standard TCP/IP sockets. Once a MadKit kernel is informed of the network location and identify of another MadKit kernel, it communicates with the remote kernel directly and transfers messages from its agents to agents running in the other platform-kernel combination.

MadKit provides a unique platform dependent identifier for each agent by instances of the *madkit.kernel.AgentAddress* class. This identifier is used throughout TTAS and the TTAS APIs.

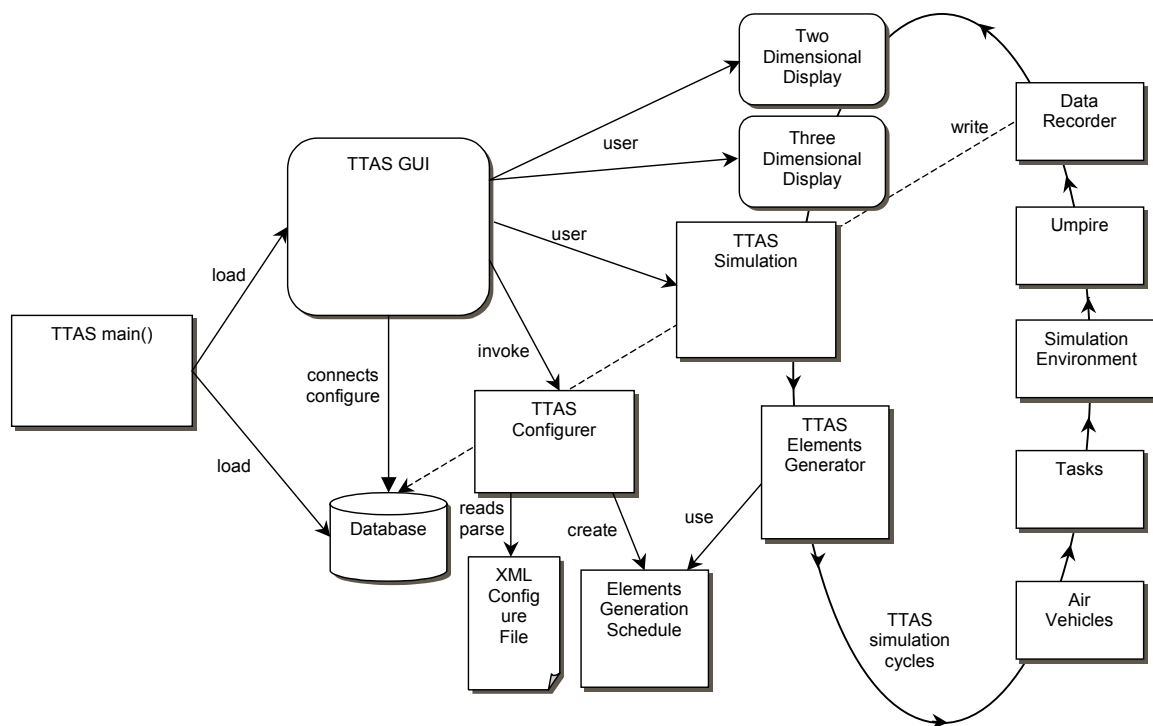


Figure 2. Testbed for Taskable Agent Systems design

MadKit manages the *agent lifecycle*, which consists of *activation*, *execution*, and *termination*. *Activation*, a call to the method *activate* (of *madkit.kernel.AbstractAgent*), registers the

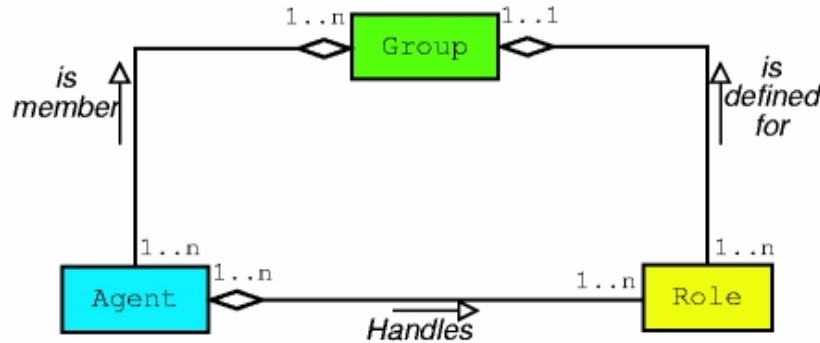


Figure 3. The Aalaadin agent organization model

TTAS agent with the MadKit micro-kernel. A TTAS agent is a Java *extension* of the *madkit.kernel.AbstractAgent* class. *Execution* is the agent performing its programmed activities within the MadKit framework, and *termination* is the cessation of those activities.

When an agent is *activated* it is registered as a member of the MadKit *community* composed of all agents running in the current micro-kernel-platform combination. Also, it can request to be a member of a *group* and can request a particular *role* in that group. MadKit may reject such requests since a developer can establish criteria for membership in a group and for playing a role. The notions of *community*, *groups*, and *roles* arose from Ferber's research into the structure of artificial organizations for MASs, the theory of which is named *Aalaadin*. Fig. 3 (taken from one of Ferber's papers) depicts a schematic representation of these notions.

TTAS defines three groups *Simulation*, *Vehicles*, and *Tasks*. The UAVs occupy the role of *AirVehicle* in both the *Vehicles* and *Simulation* groups. Ground target sites occupy the role of *task* in both the *Tasks* and *Simulation* groups. Other TTAS simulation specific agents occupy other roles and will be discussed below.

TTAS uses MadKit's *message* capabilities for all agent-to-agent communications. MadKit defines a set of messages that includes simple text messages, XML messages, Java *Object* encapsulating messages, Speech Act Messages, Knowledge Query and Manipulation Language (KQML) messages, and Agent Communication Language messages. We have extended MadKit's basic *madkit.kernel.Message* to messages appropriate to the TTAS UAV simulations, such as *Launched_AgentI*, *Kill_Agent*, *Agent_Killed*, *Site_Added*, etc. With MadKit messages can be sent to a particular agent (identified by the *madkit.kernel.AgentAddress*) instance, broadcast to all agents that occupy a specific *role* within a *group*, or broadcast to all activated agents.

MadKit agents can be constructed such that each agent executes asynchronously within its own thread or such that all agents execute synchronously (sequentially) in a single thread. Our design uses the synchronous approach (MadKit synchronous engine) so that competition for computing resources is not a problem. Such competition can occur in the JVM with many (greater than 10) single threaded simultaneously execution agents, which is the case for OEF coordination and adaptation design experiments.

4.2.2 TTAS *main()*Method

As with all Java programs TTAS is invoked from its *main* method. This method loads the

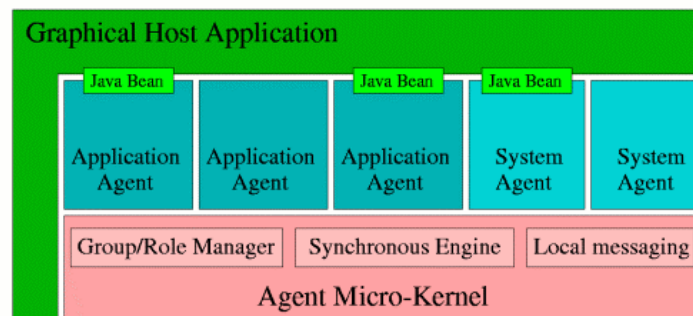


Figure 4. The MadKit architecture (from Ferber)

handler for data recording indicated by the experimenter on the command line, loads the TTAS' Graphic User Interface (GUI), and places a reference to the data recording handler in the GUI. As discussed below, even though all data recording takes place through Java Database Connectivity (JDBC) different vendor's database may require slightly different invocation and connection procedures. Loading the data recording handler at run time allows TTAS the flexibility to handle these situations without re-implementation for each different database.

4.2.3 TTAS GUI

The TTAS GUI is implemented with the Java Swing package. It holds the two- and three-dimensional display and summary data measurement plots and provides the menus and controls for the following:

1. Selecting the TTAS configuration file for the experiment
2. Specifying the length of a time step and the duration between time steps (which must be non-zero to prevent the MadKit threads and the Java Swing execution thread from coming into conflict)
3. Starting and stopping the simulation experiment run
4. Selecting, hiding, and showing the two- and three-dimension and data displays

4.2.4 TTAS Experiment Simulation Configurer

After the experimenter selects the particular experiment configuration file TTAS reads and parses the XML file and through Java dynamic class loading and reflection (as described above) creates the classes and class instances needed to execute the simulation. All these Java elements are encapsulated within a data structure, *BasicTestBedElementsGenerationSchedule*, that is used by other elements within TTAS. The *configurer* is designed to handle general Java classes and instances so that TTAS can be extended to include additional features such as *threats* or *terrain* without re-implementation of this important component.

4.2.5 TTAS Simulation Component

The TTAS simulation component *TestBedSimulation*, which is extension of the MadKit synchronous engine class *madkit.kernel.Scheduler*, provides the connection to the MadKit micro-kernel necessary for execution of the simulation. The simulation component is a *threaded* MadKit agent that is managed by the MadKit micro-kernel but executes within its own thread. Within *TestBedSimulation* a *loop* (a Java *while* control loop) cycles until stopped by the experimenter. Within this loop *drivers* for the groups *Vehicles*, and *Tasks* and several other agents (such as the two- and three-dimension displays) are called during each cycle. A *driver* is an extension of the *madkit.kernel.Activator* and is a part of the MadKit synchronous engine. Each TTAS MadKit agent that is a member of a particular group (or is to be called by its own driver) implements a method that will be called at each time step. For the *Vehicles* group the method is *moveon*, for the *Tasks* group it is *update*, for the two- and three- dimension displays it is *observe*, and so on. The driver for each group (or agent) calls that method for all the agents of that group. MadKit through Java reflection provides the driver with a reference to every agent that is *currently* a member of its group since agents can be created and destroyed or can join and leave groups as the simulation proceeds. Additionally, the simulation component creates and joins the TTAS groups (in the role *startUp* or *Scheduler*), creates the drivers and registers them with the MadKit micro-kernel, and creates and launches the *TTAS Elements Generator*, *Simulation Environment*, *Umpire*, *Data Recorder*, and two- and three- dimension display components, which are also MadKit synchronous agents. Refer to Fig. 2 for the simulation components looping cycle.

4.2.6 The TTAS Elements Generator Component

We have designed TTAS so that agents such as UAVs and target ground sites (tasks) can be created or destroyed at anytime during the simulation on a regular schedule, randomly, or in response to requests. The *TestBedElementsGenerator* class instance that performs this function is a standard MadKit synchronous agent. At each time step it checks whether it has received any messages requesting that an agent be created or destroyed and examines data contained within this experiment's *TestBedElementsGeneratorSchedule* class instance (created by the *configurer* component as discussed in Section 3.3.4), in order to determine whether new agents should be launched. If an agent is to be killed, it removes it from the simulation (only the creator of an agent or the agent itself can destroy an agent). If a new agent is to be launched, the *TestBedElementsGenerator* extracts the appropriate class information or class instance references from the experiment's *TestBedElementsGeneratorSchedule* class

instance and uses Java reflection (as discussed previously) to create the agent and launches it into the simulation.

In response to a single request or schedule time, the *TestBedElementsGenerator* can generate an individual agent (a new UAV or task) or multiple agents with randomly selected parameters. An example of the latter case is the generation of a large number of target ground sites at the beginning of an experiment simulation run: the geo-locations, the *type*, and the *value* 50 to 100 ground sites are generated randomly.

After an agent is launched or destroyed the *TestBedElementsGenerator* sends messages to other agents informing them of the event. For example, the two- and three- dimension displays use these messages to provide graphic representations and annotations. Each UAV agent is notified when a new known ground site is added and when a completely serviced site is removed. The data recorder receives all such agent launch and kill messages.

4.2.7 Air Vehicles

An experimenter inserts agent implementations into TTAS through *Air Vehicles*. We have defined a set of APIs as Java Interfaces for Air Vehicles in the *airvehicle* Java Package. The major Air Vehicles' API Interfaces and their purposes are listed in Table I. Reference implementations of some of these Interfaces, which can be used in most OEF MAS simulations, are part of the *basicvehicle* Java Package. We designed the *Air Vehicles* part of TTAS and the reference implementations to be highly modular. For a particular physical air vehicle, such as a vertical-takeoff-landing (VTOL) hoverable UAV, only the *agent implementation* that controls the air vehicle needs be changed to investigate a different approach to coordination and adaptation. As discussed above, with respect to TTAS this change is isolated to the modification of a few lines in the configuration file. Of course, the new agent must implement the Java Interfaces identified in Table 1. Only Java primitives and classes, MadKit classes, and TTAS classes are referenced in any of the APIs.

TTAS air vehicles are MadKit agents and must extend the *madkit.kernel.AbstractAgent* class as well as implement TTAS' *Vehicle_I* interface (and the *madkit.kernel.ReferenceableAgent*, which is necessary for the agent to execute within MadKit's synchronous engine). Fig. 5 displays schematically the layout of the *BasicVehicle* reference implementation. We have designed and implemented a

Interface Name	Purpose
----------------	---------

Table 1. The TTAS APIs for Air Vehicles.

Vehicle_I	Defines methods TTAS needs to gain access to internal parameters of an air vehicle. Includes the <i>moveOn</i> method invoked by the <i>Vehicles</i> driver (<i>Activator</i>) and methods to retrieve references to the physical state of the vehicle, the environment surrounding the vehicle, the plan the vehicle is following, and the agent controlling the vehicle. A Java class for an Air Vehicle (UAV) is a Java
-----------	--

	<i>extension</i> to MadKit's <i>AbstractAgent</i> class that <i>implements</i> this Interface. A reference implementation that can be used by most MAS is <i>basicvehicle.BasicVehicle</i> .
VehicleState_I	The <i>physical state</i> of the <i>Air Vehicle</i> is encapsulated within implementations of this Interface. The quantities that must be part of the physical state are a unique identifier object (usually the MadKit generated <i>AgentAddress</i> instance), the three-dimensional position, velocity, and acceleration, the simulation elapsed time at which this state was valid, the three-dimensional orientation of the Air Vehicle (with respect to vehicle's center), the current navigational mode, and an indicator of whether or not the vehicle has undergone a collision. The reference implementation is <i>basicvehicle.BasicVehicleState</i> .
VehicleAgent_I	Defines the TTAS API for the actual <i>agent</i> that is the subject of research. Implementations are encapsulated within the <i>Air Vehicle</i> , such as <i>BasicVehicle</i> . Defines the method <i>update</i> that must be called at each time-step and the parameters that are part of the call. Also defines a method of retrieving and sending messages since the Air Vehicle itself is the MadKit agent that receives and sends messages. No reference implementation.
nullAgent_I	An extension to <i>VehicleAgent_I</i> that can be used to handle <i>autonomic</i> responses, that is, reactions to situations in the environment that should/can be taken immediately without invoking the complete intelligence of a full-blown agent. Collision and threat avoidance and waypoint-based navigation are examples. Intended to encapsulate quick-response survival and locomotion behaviors. It is possible to isolate knowledge of the kinematics of the Air Vehicle necessary for locomotion and navigation here. No reference implementation.
VehicleParameters_I	Defines retrieval methods for the <i>performance</i> parameters of the Air Vehicle, which are used by the <i>VehicleController_I</i> and may be used by the <i>nullAgent_I</i> . Includes maximum acceleration, deceleration, climb rate, descent rate, and speed; minimum and cruise speeds; mass; dimensions; and type. No reference implementation.
VehicleConstraints_I	Defines retrieval methods for sensor and communication distances. Task (ground site) sensor constraints can depend upon the type (Java class) of task and the speed of the air vehicle. Maximum awareness and communications distances to other air vehicles and the minimum distance allowed between vehicles can depend upon the relative closing speed. No reference implementation.
VehicleEnvironment_I	Defines the methods for accessing the position of other air vehicles and of tasks consistent with the data encapsulated within the <i>VehicleConstraints_I</i> implementation and referenced relative to the current air vehicle of interest. A reference implementation is <i>basicvehicle.BasicVehicleEnvironment</i> .
VehicleCommand_I	Indicates a navigation or service command. Can be issued by the <i>Agent_I</i> or <i>nullAgent_I</i> implementations. Navigation commands are interpreted by the <i>VehicleController_I</i> implementation and service commands by the <i>VehicleService_I</i> implementation. Reference implementations for acceleration and deceleration, ascent and descent, and heading changes.
VehicleCommandLoad_I	A data structure into which <i>VehicleCommand_I</i> instances can be placed and retrieved. <i>basicvehicle.BasicCommandLoad</i> is a

	reference implementation as a FIFO queue.
VehicleController_I	Defines the method <i>update</i> that must be called at each time step to move the air vehicle. Translates the navigational commands into simulated physical motion through use of the kinematics model implemented within it and the parameters of the <i>VehicleParameters_I</i> implementation. Reference implementations for fixed wing and for hoverable air vehicles.
VehiclePlan_I	If an air vehicle is to follow a pre-generated or real-time generated plan, the plan must implement this Interface. No reference implementation, but a test implementation based upon waypoints is available.
VehiclePlanState_I	Intended to capture the current state of the execution of a <i>VehiclePlan_I</i> . No reference implementation, but a test implementation based upon waypoints is available.
VehiclePlanManager_I	Manages a <i>VehiclePlan_I</i> implementation. No reference implementation, but a test implementation based upon waypoints is available.
VehicleService_I	An interface for the management of <i>sensors</i> or of a component that <i>services</i> a task in some manner. No reference implementation, but several test implementations are available.
VehicleServiceState_I	Captures the current state of a sensor or service component. . No reference implementation, but several test implementations are available.
VehicleControlsGUI_I	If used or enabled provides visual access to at least the physical state of the air vehicle. Can be used to display the state of the agent in control of the air vehicle or, with the implementation of controls, to control the air vehicle and its parameters. No reference implementation.

VTOL, hoverable air vehicle kinematics model (the OEF UAV- β) for the *nullAgent_I* and *VehicleController_I* APIs within the *BasicVehicle* implementation. For test purposes, we use the parameters of the Moller Skycar (<http://www.moller.com/skycar/>). A primary assumption of our implementation is that only *straight-line* (rectilinear) motion occurs during a single time step, which implies the following:

- Heading changes applied at beginning of time step
- Velocity at end of time step = acceleration $\times$ time step length ($v = a\Delta t$)
- Position at end of time step = beginning position + velocity at beginning of time step $\times$ time step length + $\frac{1}{2} \times$ acceleration $\times$ time step length² ($p(t_0 + \Delta t) = p(t_0) + v\Delta t + \frac{1}{2} a\Delta t^2$)

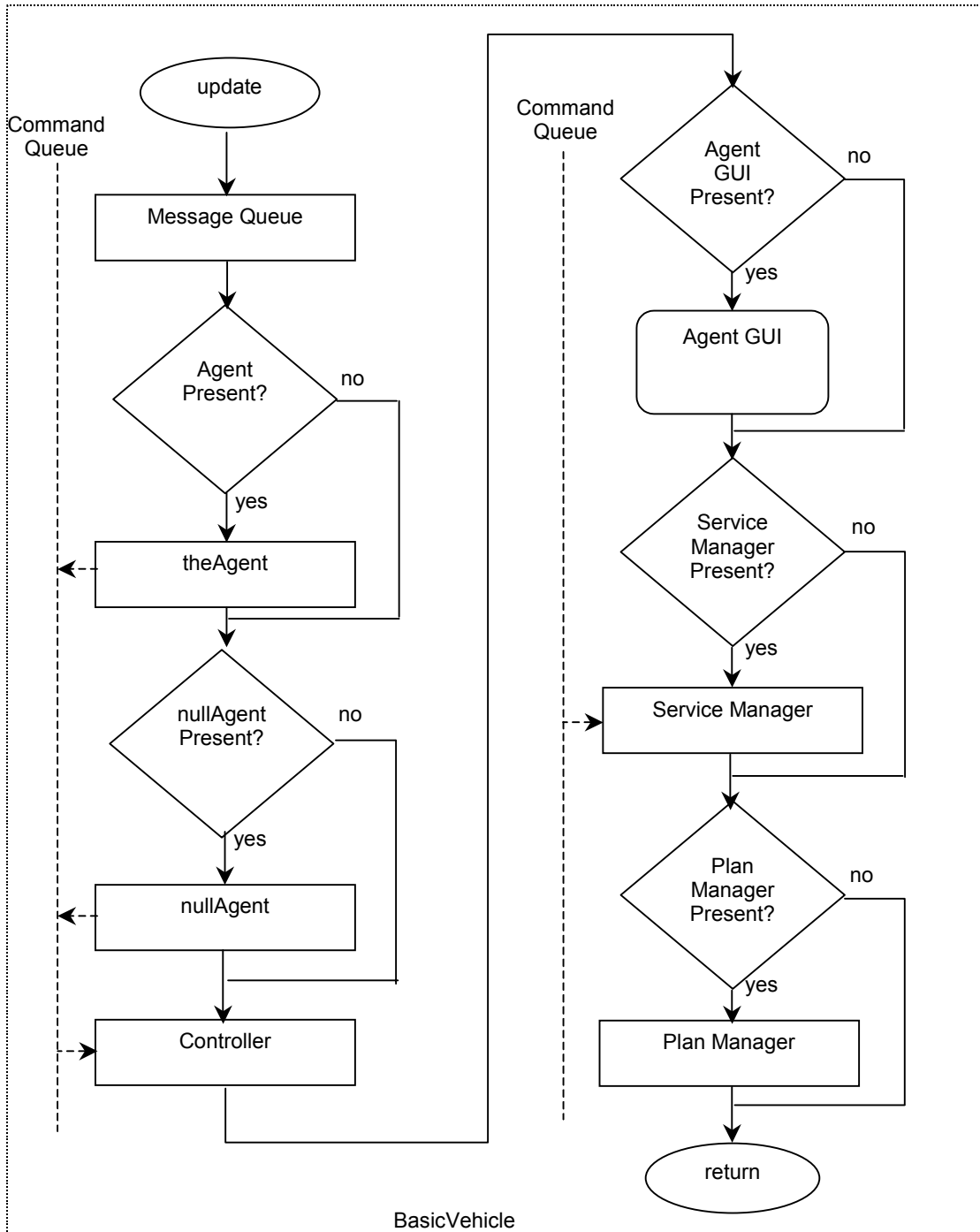


Figure 5. The basic vehicle reference implementation outline

The *nullAgent_I* implementation navigates between *waypoints* (a waypoint is a position, an optional time, and an optional velocity) where target ground site (tasks) locations are denoted by waypoints and has several navigation modes (CRUISE_LEVEL_FLIGHT, ASCENDING, DESCENDING,

ZERO_VELOCITY_Z, ACCELERATING_XY, DECELERATING_XY, ZERO_VELOCITY_XY, STATIONARY, and DECELERATING_XY_TO_ZERO_VELOCITY_XY).

4.2.8 Tasks

TTAS Java Package *job* contains the *Tasks* APIs defined as Java Interfaces. Analogous to *Air Vehicles* we created reference implementations for OEFexperiments. The *Tasks* Interfaces and purposes are listed in Table 2. The target ground site class of the OEF is

Table 2. The TTAS APIs for Tasks.

Interface Name	Purpose
Task_I	Indicates to TTAS that this object is to be treated as a <i>task</i> within the simulation and provides methods to retrieve the internal parameters of the task. Defines the method <i>update</i> that is called at each simulation time step, which is when the implementation performs computations. No reference implementation but two test implementations: (1) <i>WaterTank</i> , which realizes the simple notion of a task with capacity that can be serviced simultaneously (or sequentially) by multiple agents and (2) <i>TargetTask</i> , which are the ground sites of the OEF.
TaskState_I	Defines methods to retrieve the parameters of the <i>physical state</i> of the task. The quantities referenced are a unique identifier (usually the MadKit generated <i>AgentAddress</i>), the three-dimensional position, the simulation elapsed time at which this data was valid, an indicator of whether or not the task is completely serviced, and the simulation elapsed time at which this task was serviced. <i>job.TaskState</i> is a reference implementation. <i>WaterTankState</i> and <i>TargetTaskState</i> correspond to the test implementations described above.
TaskCharacteristics_I	Implementations have methods to retrieve parameters that characterize a task. No reference implementation. <i>WaterTankCharacteristics</i> specifies capacity, maximum service rate, and physical dimensions. <i>TargetTaskCharacteristics</i> specifies the task <i>type</i> (as specified in the OEF one of A...F) and <i>value</i> .
TaskCharacteristicsRanges_I	Implementations provide methods for retrieving the extremes of the parameters encapsulated within <i>TaskCharacteristics_I</i> implementations. Used by the <i>TestBedElementsGenerator</i> to generate random values for these parameters.
Connector_I	Defines methods for a servicing agent to connect to a task. No reference implementation. <i>WaterHose</i> is the connector for the <i>WaterTankTask</i> . (The <i>WaterPump</i> service manager in an air vehicle connects to a <i>WaterTank</i> through the <i>WaterHose</i> , for example.
TaskServiceCommand_I	Indicates that the class should be interpreted as a command that concerns servicing a task. No reference implementation.

an extension of the `madkit.kernel.AbstractAgent` class and an implementation of the `job.Task_I` and `madkit.kernel.ReferenceableAgent` interfaces.

4.2.9 The TTAS Simulation Environment Component

The TTAS Simulation Environment component, `nullAgent.util.EnvironmentObserver`, updates each Air Vehicle's knowledge of its locale (the `VehicleEnvironment_I` implementation such as the `BasicVehicleEnvironment` reference implementation) consistent with the constraints encapsulated within the `VehicleConstraints_I` implementation.

`EnvironmentObserver` is an extension to the MadKit class `madkit.kernel.Watcher` that was references to registered `madkit.kernel.Probe` instances, which hold references to all the MadKit agents of a specific *group* and *role*. For example, `EnvironmentObserver` retrieves the positions and velocities of each Air Vehicle, checks which are within the maximum visibility constraints to the Air Vehicle whose locale is being updated, and performs the transformations necessary to reference to this Air Vehicle the positions and velocities of those that are visible. The transformations use the three- and four-dimensional matrix transformation classes of the Java 3D Package `javax.vecmath`. Air Vehicle centric information about tasks is updated similarly. Since `madkit.kernel.Watcher` is a MadKit agent, so is `EnvironmentObserver`, which is why it is called with the `TestBedSimulation` looping cycle.

4.2.10 TTAS Umpire Component

The *Umpire* performs several functions. It handles simulation-, group-, and/or role-wide message broadcasts; it can verify completion of task service and dispense rewards; and it can enforce the rules-of-the-game. As with the `EnvironmentObserver` class *Umpire* is an extension of the MadKit `madkit.kernel.Watcher` class and through the registered `madkit.kernel.Probe` instances has access to all agents of specified *groups* and *roles*. It also is a MadKit agent.

4.2.11 TTAS Data Recorder

The `StorageWatcher` (the Data Recorder component), when enabled, writes all messages exchanged between *air vehicles* and *tasks* (if any) in the relational database through the JDBC connection established at simulation startup. The relational schema into which data is recorded is created at execution time, which implies that each experimental data set is a separate database instance. The current implementation (when enabled) records the *physical state* of each air vehicle and its *local environment* at each step. The `StorageWatcher` is also an extension of the MadKit `madkit.kernel.Watcher` class. Redesign underway to record user specified data at specified time intervals.

We use the Mckoi SQL Database, which is Open Source and implemented in Java. The Mckoi database can be embedded within TTAS, but TTAS execution is slowed as a single JVM handles both TTAS and the database. Generally we use Mckoi as a standard (remote) database server that executes within its own JVM. In this configuration performance is acceptable even when TTAS and Mckoi are on the same computer. As noted in Section 3.3.2, since all database access occurs through JDBC, any database can be used with suitable implementations of the TTAS API (Java Interface) `JDBCStorageHandler_I`.

4.2.12 Two Dimensional View

The Two Dimension Display provides a downward looking, *wizard's* view of the simulation world. It displays the locations of air vehicles and tasks; can change a task's color after the task is serviced; and can display air vehicle and task identification. A coordinate grid is displayed divided as appropriate to the simulation world size into units of 1, 2, 5, or 10 kilometers. The display is implemented in Java Swing with Java graphics as an extension to the

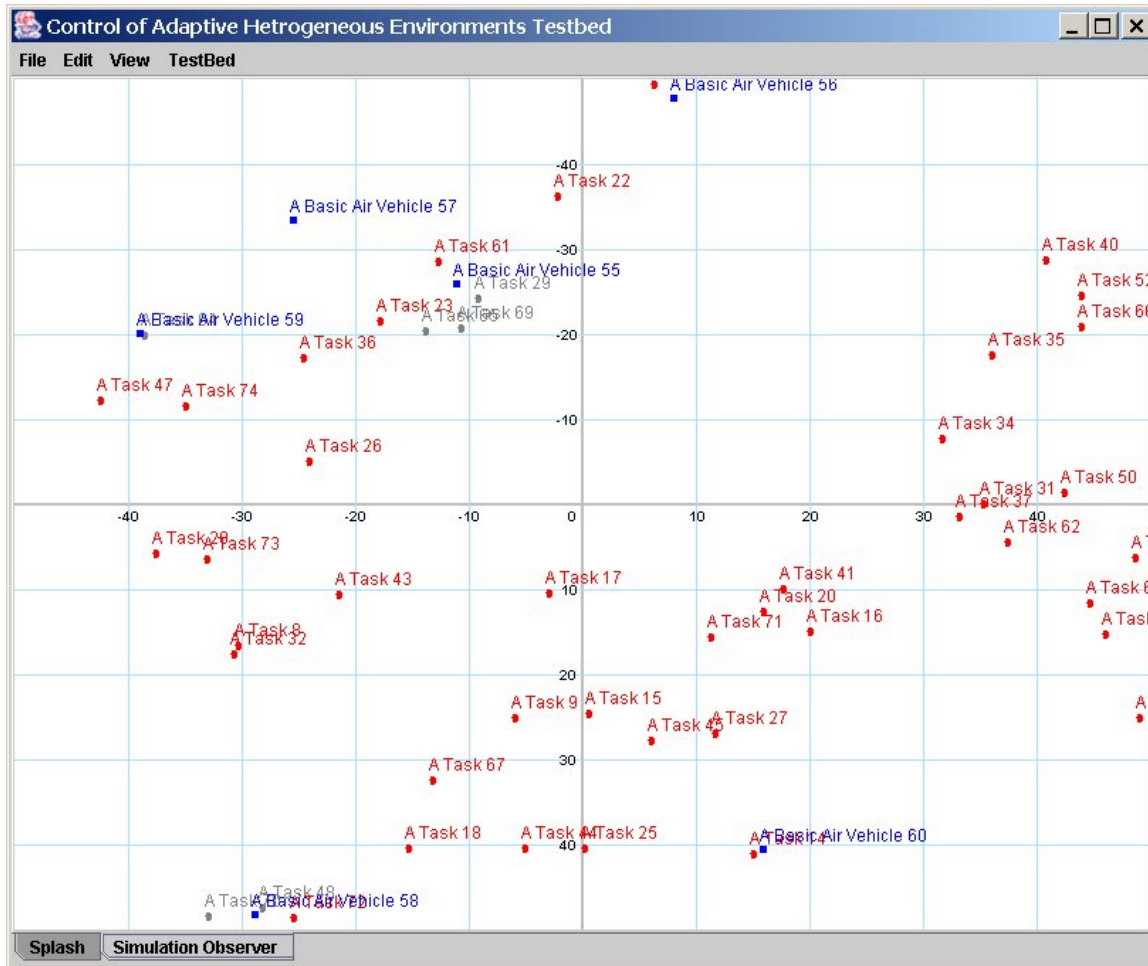


Figure 6. TTAS' two-dimensional display

madkit.kernel.Watcher class. In general this display is intended for debugging MAS implementations and experiment configurations. Our plans include extending the Two Dimensional View to include terrain and other geographic features such as road and buildings. A screen shot of the current Two Dimensional View is displayed in Fig. 6.

4.2.13 The Three Dimensional View

The Three Dimensional View provides a true three dimensional view of the locale surrounding the *camera* position. The camera position can be selected to be the position of a specific task and can be rotated through all three angles and moved through all three linear dimensions via mouse

movement. The Three Dimensional View is Implemented in Java 3D as an extension to the MadKit's *madkit.kernel.Watcher* class. Planned extensions include zoom, terrain, geographic features, and views from a UAV, which will be quite interesting when terrain is included in the simulation.

4.3 Milestones and Future Directions

4.3.1 Milestones

During this period of performance we developed a working implementation of the Testbed for Taskable Agent Systems that supports experiments on Multi-Agent Systems of UAVs collaborating to find and survey ground targets. Different approaches to coordination and adaptation can be investigated quantitatively by simply replacing the agents that control the UAVs. Standard experiment configurations are archived in a reusable XML-based configuration file. Post-experiment analysis is made possible by extensive data recording to a relational database. A complete set of APIs is provided for air vehicles and tasks. Reference implementations of these APIs are provided for UAVs and target ground sites.

4.3.2 Future Directions

Complete enhancements necessary to support all OEF design problems, such as moving targets, unreliable information, cross mission tasking, and time critical targeting. Include terrain and geographic features and the effects of terrain on line-of-sight sensing and communication. Test proposed OEF experiment configurations and establish baseline measurements for these configurations. Test kinematics models for fixed wing UAVs. Use a simulation platform for Alphatech's research. Work with TASK PI organizations in order to perform experiments in TTAS on their approaches to the OEF design problems.

5 Cooperative Learning and Dynamic Control MAS Research

Our work has been focused on Multi-Agent System (MAS) problems in which each agent is required to make a series of online decisions based on its belief about the state of its environment. Each agent is an independent and distributed participant in the MAS, whose purpose is to maximize the MAS performance through local action. By developing new technologies for coordination and adaptation within the context of a MAS formulation of a UAV problem, our work will help facilitate advances in DoD capabilities in surveillance, targeting, biological/chemical monitoring, and battlefield command and control, just to name a few. These new capabilities for long-term, autonomous UAV missions will dramatically increase the DoD's information superiority, while simultaneously reducing the risk for loss of human life.

As depicted in Figure 7, in our formulation each agent decision results in the execution of an action, the result being a transition to a new state and a real-value reward reflecting the immediate value of the new state. We are interested in MAS problems where the agents are

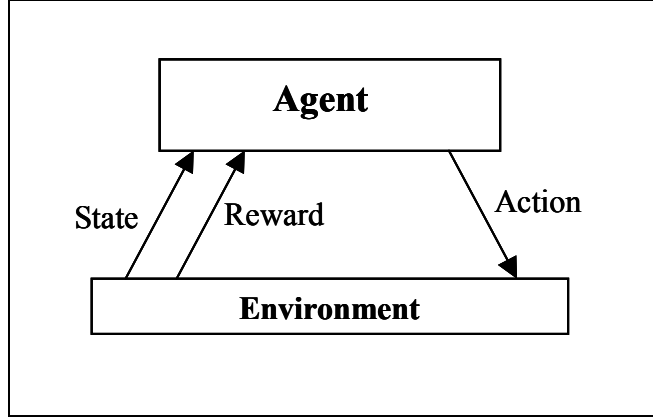


Figure 7. Interaction between the agent and its environment in a Reinforcement Learning context

learning about their environment incrementally. We further characterize this as Reinforcement Learning by limiting the agents to a feedback value that reflects the immediate efficacy of an action as a function of the state transition that it invokes, but not an omniscient response indicating the correct action for the current state, as would be the case with a Supervised Learning approach.

When using Reinforcement Learning for control, the objective is generally to learn a value function for either states or state-action combinations such that a policy function, $\pi(s)$, can be defined in terms of the value function and used to compute the action that the agent will execute in any state s . From dynamic programming, we know that one general method for optimal policy computation is as follows:

$$policy^*(s) = \underset{a}{argmax} \sum_{s'} \mathcal{P}_{ss'}^a V(s') \quad (1)$$

Where $V(s')$ is the expected value of state s' .

Although there are well-known Reinforcement Learning algorithms that are suitable for many Markov Decision Processes, learning the value functions can be computationally challenging as the complexity of the environment grows. By the law of large numbers, Reinforcement Learning algorithms are proven to converge to optimal policies if they systematically explore the totality of the state and action space infinitely often. Large state spaces or those with continuous attributes, as well as large action spaces, can all contribute to the complexity of the environment and intractability of obtaining good solutions.

5.1 Objective

Our research objective has been to allow heterogeneous agents operating as a Multi-Agent System to each benefit from the collective learning capabilities of the entire group. Specifically, we would like our agents to be capable of exchanging and exploiting learned knowledge. We expect to achieve several benefits from knowledge sharing.

- **Divide and Conquer** - Large, complex state spaces or large action spaces can all cause the computational demands of the learning problem to explode. Multiple agents each attacking different parts of the same problem will reduce these demands for individual agents.
- **Expanding Capabilities** - Intuitively, heterogeneous agents will often have heterogeneous capabilities, in terms of perception or actions, which correlate to the type or fidelity of the agent's observations about the environment. An ability to exchange the subsequently derived knowledge would allow all agents to take advantage of the aggregate of the group's capabilities.
- **Credit Assignment** - A problem that frequently complicates reinforcement learning applications is that of assigning appropriate credit to states or actions for their contribution to achieving goals. The sequential experiences encountered by an agent make it difficult to separate the salient interactions from those that are coincidental. The iterative exchange of learned knowledge may provide each agent with multiple points of reference for determining true causality.
- **Improving Quality** - Related to the desire to reduce the computational demand of learning in large spaces, we would also like to have the agents achieve beneficial performance more quickly. Allowing agents to benefit from the mistakes of others will reduce the collectively accumulated penalty for incorrect actions.

5.2 Reinforcement Learning

Reinforcement learning can best be thought of as a classification of a learning problem [3, 8], rather than a specific solution. Reinforcement learning algorithms iteratively derive a value function for states or state-action pairs. Any algorithm that accomplishes that goal can be classified as a reinforcement learning algorithm. Reinforcement learning differs from traditional supervised machine learning in that there is no omniscient supervisor that provides the agent with examples of optimal behavior. Instead, the learner relies on iterative rewards that indicate the efficacy of an action or sequence of actions in terms of the value of the states to which the agent is transitioned. It also differs from unsupervised learning in that there is some external feedback with which the learner can measure its efficacy, where unsupervised learners rely on self-assessment to measure their progress.

Exploitation vs. exploration is an important consideration in the formulation of a Reinforcement Learning solution. To learn the value of states and actions, an agent must explore its range of actions over the range of the state space. To accomplish this, the agent must at times act against what it believes is the best course of action. The agent must eventually arrive at some synergy

between exploration and exploitation to act successfully in the environment. The intractability of exploring large environments is a key motivation for this work.

SARSA(λ)

The SARSA(λ) algorithm [8] serves as the model-free Reinforcement Learning algorithm for our initial research. The SARSA algorithm derives its name from the quintuple $\{s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\}$ (*state-action-reward-state-action*). SARSA is considered an on-policy control algorithm, since the policy that's being continuously updated is simultaneously used for control. The algorithm is model-free since it operates with no model of the environment dynamics. The use of an eligibility trace $e(s, a)$, to propagate a portion of the reward back through the sequence of state-action pairs that allowed the agent to arrive at the reward state is particularly effective in

```

Initialize  $Q(s,a)$  arbitrarily and  $e(s,a)=0$ , for all  $s, a$ 
Repeat (for each episode):
  Initialize  $s, a$ 
  Repeat (for each step of episode)
    Take action  $a$ , observe  $r, s'$ 
    Choose  $a'$  from  $s'$  using policy derived from  $Q$ 
     $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$ 
     $e(s, a) \leftarrow e(s, a) + 1$ 
    For all  $s, a$ :
       $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$ 
       $e(s, a) \leftarrow \gamma \lambda e(s, a)$ 
     $s \leftarrow s'; a \leftarrow a'$ 
  until  $s$  is terminal

```

Figure 8. Tabular SARSA(λ) online learning and control algorithm

speeding up the convergence of the SARSA(λ) algorithm.

5.2.1 Cooperative Learning

The algorithm in Figure 8 is the tabular form of SARSA(λ), meaning that it is suitable for discrete state spaces where each state variable takes on a relatively small number of values, such that $Q(s,a)$ can be fully enumerated in matrix form. The set of states, S , and actions, A , are developed in accordance with classic Markov Decision Process formulations, discussed in a wide array of literature, and only affect the SARSA(λ) algorithm and our proposed work to the degree that they impact the scalability of the algorithm itself.

Although the primary focus of this work is to incrementally assess the quality of an agent's evolving knowledge, it should also be noted that under some circumstances knowledge representation would play a key role in knowledge sharing. Part of our research objective can be

viewed as an attempt to develop a method by which agents can parallelize the process of learning about the environment. Like most efforts to parallelize algorithms, the challenging aspect is the reintegration of the n partial solutions. Some heterogeneous agent systems will inevitably have varied sub-goals, unique value/cost functions, and different perception and action capabilities. Clearly these attributes will all affect the internal knowledge representation of the agent, causing the knowledge held to be *situational*, severely complicating any exchange and integration of knowledge. Therefore a generalization mechanism must be employed for agents operating under these circumstances. Relevant work has been accomplished in agent communication languages and knowledge interchange formats, but substantial challenges remain with regard to the impact of learning and local valuation in heterogeneous agent systems. In addition to this challenge, the fact that an agent will now have multiple perspectives on which to base decisions introduces a totally new quantity of information. There is significant research to be done to address the meaning that can be derived from the comparison of multiple models learned in the same environment and to define a suitable set of actions that the agent can use to exploit this information. Actions may include the ability to exchange deeper information to elucidate the motivation for a learned function or to further constrain or expand exploration processes.

5.3 Quality of Knowledge

One of the first questions we think about in terms of agents exchanging their learned functions is, should agents exchange all information or just good information and how do we measure goodness? In our context, good information is a Q-function value in which the agent has a high degree of confidence, where the measure of confidence reflects the convergence of the estimated value to the true value. The SARSA(λ) algorithm is a process used to repeatedly refine an estimate of the value of executing a specific action in a specific state. So at any given time, we'd like our agent to be able to determine how confident it is that the estimated value is close to the actual value.

Our approach is to incrementally apply statistical confidence interval measurements to this problem, which are widely addressed in inferential statistics literature. The general idea behind our approach is that the learned Q-values for state-action pairs, $Q(s,a)$, will be relatively volatile while the values are still in the process of converging to their true values under the current policy. By establishing a threshold within which we determine the true value must lie with respect to the running estimate and establishing a confidence measure with which we require the measurement to adhere, we can evaluate the agent's belief about the quality of its learned knowledge. It is important to note that the quality value we are measuring is the stability of a learned value under a given policy, π . This has implications with respect to changing policies during the course of learning, generally done to reduce exploration in favor of exploitation. We will discuss these implications in the following sections.

5.3.1 Quality Measurement

In this section we discuss our quality measurement approach in detail. We assume the reader's familiarity with basic statistical measures, such as the mean and standard deviation.

To calculate the confidence metric, we'll need to derive a few statistical measures of the Q-function distribution, the first being the standard error [2, 5], defined in Equation 2:

$$se(\bar{\theta}) = \frac{\sigma}{\sqrt{n}} \quad (2)$$

Where θ is the parameter being estimated, σ is the standard deviation of the sampled parameter values, and n is the sample size. The standard deviation can be computed using Equation 3.

$$\sigma^2 = \frac{\sum x^2}{n} \Rightarrow \sigma = \sqrt{\frac{\sum x^2}{n}} \quad (3)$$

Where $\sum x^2$ is the sum of the squared individual deviations, n is the sample size, and σ^2 is the variance. Formally, we are estimating the standard deviation so the use of σ , which typically refers to the true value, is incorrect. Henceforth all references to the standard deviation are references to the estimated value, which will be represented by S .

This method of computing the standard error isn't suitable for our approach because our agents are collecting samples sequentially, whereas Equation 3 is meant to be applied to a static sample set. Saving all data points to recompute the standard deviation isn't computationally feasible, however Equation 4 can be used to incrementally compute a running average for any parameter θ . Since the standard deviation is an average of individual deviations, Equation 4 can be readily used for our purposes.

$$F_{t+1}(\theta) = F_t(\theta) + \frac{1}{t+1} [\theta_{t+1} - F_t(\theta)] \quad (4)$$

Where θ is the parameter for which the average is being calculated, θ_t is the observed value of the parameter at time t , and $F(\theta)_t$ is the computed average of θ at time t . Equation 4 assumes that one observation of θ is received at each timestep t .

Normally one computes the standard deviation by averaging the distance between each sample point and the Central Tendency, with the mean being the most common measure of Central Tendency. In our case, since we are applying these calculations to the SARSA(λ) algorithm, the work of estimating the Central Tendency of each $Q(s,a)$ is being done for us by the algorithm. Therefore, we can use the current estimate, $Q_n(s',a')$ as the measure of Central Tendency, where n is the n th time action a' has been executed in state s' . Equation 5 illustrates the application of Equation 4 to the SARSA(λ) parameters.

$$S_{t+1}(s', a') = \sqrt{S_n^2(s', a') + \frac{1}{n+1} [(Q_n(s', a') - Q_{n+1}(s', a'))^2 - S_n^2(s', a')]} \quad (5)$$

Where $S_n^2(s', a')$ is the variance of $Q_n(s', a')$ after the n th time action a' has been executed in state s' .

Using Equation 5, we are able to fold the deviation of the current value, $Q_{n+1}(s',a')$, into the running standard deviation.

Although Equation 5 provides us with an efficient means of calculating the standard deviation and therefore the standard error, it may be desirable to attenuate the affect of older data points on the running calculation of the standard deviation, since the learned values, Q_t , should generally be getting closer to the true values, Q^* .

To do this, we would normally compute a moving average over a sliding window of the past m data points, where $m \perp n$. However, the incremental approach illustrated in Equation 5 does not lend itself to a sliding window implementation because there is no way to extract the influence of historical data from the running statistic. Storing the past m data points and recomputing the measurement is also an unappealing option for obvious reasons. An alternative is to use exponential smoothing, illustrated in Equation 6, to compute the running average.

$$F_t(\theta) = \omega\theta_t + (1 - \omega)F_{t-1}(\theta) \quad (6)$$

Where ω is a weight parameter between $[0,1]$ that controls the emphasis on old versus new data points, θ is the parameter for which the average is being calculated, θ_t is the observed value of the parameter at time t , and $F_t(\theta)$ is the computed average of θ at time t .

Exponential smoothing is a process used in time-series analysis to incrementally compute a weighted average, with the influence of older data points on the mean being attenuated exponentially. The expanded form in Equation 7 depicts this point.

$$F_t(\theta) = \omega [\theta_t + (1 - \omega)\theta_{t-1} + ((1 - \omega)^2)\theta_{t-2} + ((1 - \omega)^3)\theta_{t-3} + \dots] \quad (7)$$

Where θ_t is the observed value of the parameter θ at time t , and $F_t(\theta)$ is the computed average of θ at time t .

In time-series analysis, exponential smoothing is typically used for prediction. Although we would not use it for that purpose, using it to maintain the standard deviation may cause the standard error to more accurately represent the agent's current knowledge. Although our preliminary experiments with the exponential smoothing technique showing promising results, we have not yet validated the applicability of the method. Exponential smoothing is meant to be applied only when the sample data points are independent, which in a practical sense means that each point conveys an equal amount of information to the estimation process. It is not clear that this is the case during the volatile stages of the learning process. Our future work will address this in more detail.

Equation 8 illustrates our adaptation of the exponential smoothing function to incrementally compute the standard deviation of the Q-function estimate.

$$S_{n+1}(s', a') = \sqrt{\omega [Q_n(s', a') - Q_{n+1}(s', a')]^2 + [(1 - \omega)S_n^2(s', a')]} \quad (8)$$

Now that we have a means by which we can incrementally compute the standard error, we're left with the question of how to assess the agent's confidence in its current estimate of $Q_n(s', a')$. More specifically, we want to answer the question, "are we $\phi\%$ confident that the true value under the current policy, $Q^*(s', a')$, is within some acceptable range, Δ , of the current estimate, $Q_n(s', a')$?" To answer this question we can compute a confidence interval using the statistics we've derived thus far. More precisely, we will determine how low the standard error must be in order for the agent to be $\phi\%$ confident that the estimate falls within the required range.

For illustrative purposes, we will arbitrarily say that the agent must be 95% confident that the estimate is within $Q_n(s', a') \pm \Delta$, which we refer to as a $100(1-\alpha)\%$ confidence interval, where $\alpha=0.05$. This is illustrated in Equation 9.

$$P [Q_n(s', a') - \Delta \leq Q^\pi(s', a') \leq Q_n(s', a') + \Delta] = (1 - \alpha) = 0.95 \quad (9)$$

The 95% confidence interval is a commonly adopted threshold in statistical applications, although any threshold can be computed with the same process.

Since both the central tendency, Q_n and the standard deviation, S , are estimates, we will use the T-distribution to compute the confidence interval [5] on Q^* .

Equation 10 represents the density interval of interest and is plotted in Figure 9.

$$P [-t_{\frac{\alpha}{2}} \leq T_\gamma \leq t_{\frac{\alpha}{2}}] = (1 - \alpha) \quad (10)$$

All that is left is to convert back into $SARSA(\lambda)$ terms and algebraically isolate our parameter of interest:

$$P \left[-t_{.025} \leq \frac{\bar{X} - \mu}{S/\sqrt{n}} \leq t_{.025} \right] = 0.95 \quad (11)$$

$$\Rightarrow P [\bar{X} - t_{.025}S/\sqrt{n} \leq \mu \leq \bar{X} + t_{.025}S/\sqrt{n}] = 0.95 \quad (12)$$

$$\begin{aligned} \Rightarrow & \text{Substitute: } \bar{X} = Q_n(s', a'), \mu = Q^\pi(s', a') \\ \Rightarrow & P [Q_n(s', a') - t_{.025}S/\sqrt{n} \leq Q^\pi(s', a') \leq Q_n(s', a') + t_{.025}S/\sqrt{n}] = 0.95 \end{aligned} \quad (13)$$

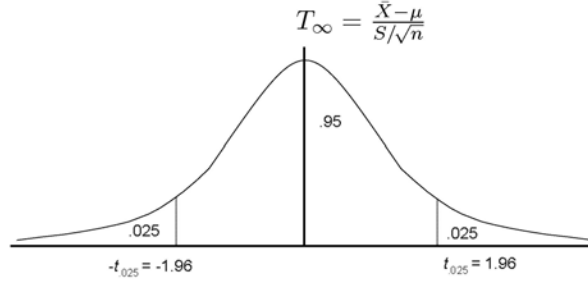


Figure 9. As the degrees of freedom approach infinity, the T distribution converges to the Standard Normal curve.

Since our real goal is to determine a bound on the standard error of $Q_n(s', a')$, from Equations 9 and 13:

$$\begin{aligned} \Rightarrow Q_n(s', a') - t_{.025} S/\sqrt{n} &= Q_n(s', a') - \Delta \\ \Rightarrow Q_n(s', a') + t_{.025} S/\sqrt{n} &= Q_n(s', a') + \Delta \\ \Rightarrow \frac{S}{\sqrt{n}} &\leq \frac{\Delta}{t_{.025}} \end{aligned} \quad (14)$$

$$\Rightarrow S \leq \frac{\sqrt{n}\Delta}{t_{.025}} \quad (15)$$

We can now see from Equation 14 that we have a bound on the standard error that provides us with 95% confidence that $Q^*(s', a')$ is within Δ of our estimate, $Q_n(s', a')$. The general form for any $100(1-\alpha)\%$ confidence interval is illustrated in Equation 16:

$$\frac{S}{\sqrt{n}} \leq \frac{\Delta}{t_{\frac{\alpha}{2}}} \quad (16)$$

The $t_{\alpha/2}$ term is obtained from the T-distribution table based on the confidence required by the human designer and the degrees of freedom γ , where $\gamma = (n-1)$, n being the number of data points over which the interval is computed. For reasonably large sample sizes (>120) we can simple use the standard normal values. The Δ parameter can be provided as a constant value or calculated as a specified ratio to the $Q_n(s', a')$ estimate itself. We will empirically evaluate these options as our research proceeds.

5.3.2 Experiment

We used a 100-cell grid as the environment for this work, illustrated in Figure 11. The agent is

1) Initialize $Q(s,a)$ & $\sigma(s,a)$ arbitrarily and $e(s,a)=0$, $\eta(s,a)=0$ for all s, a	*
2) Repeat (for each episode):	
3) Initialize s, a	
4) Repeat (for each step of episode)	
5) Take action a , observe r, s'	
6) Choose a' from s' using policy derived from Q (e.g. ϵ -greedy)	
7) $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$	
8) $e(s, a) \leftarrow e(s, a) + 1$	
9) For all s, a :	
10) $\beta \leftarrow Q(s, a)$	*
11) $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$	
12) If $(e(s, a) > 0)$	*
13) $\eta(s, a) \leftarrow \eta(s, a) + 1$	*
14) $S_{n+1} \leftarrow \sqrt{S_n^2(s', a') + \frac{1}{n+1} [(Q_n(s', a') - Q_{n+1}(s', a'))^2 - S_n^2(s', a')]}$	*
15) $S_{n+1}(s', a') \leftarrow \sqrt{\omega [\beta - Q(s', a')]^2 + [(1 - \omega) S^2(s', a')]}$	*
16) $e(s, a) \leftarrow \gamma \lambda e(s, a)$	
17) $s \leftarrow s'; a \leftarrow a'$	
18) until s is terminal	

Figure 10. Cooperative SARSA(λ) online learning and control algorithm

1,1 start	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10
5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10
6,1	6,2	6,3	6,4	6,5	6,6 10	6,7	6,8	6,9	6,10
7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9	7,10
8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9	8,10
9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9 100	9,10
10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9	10,10

Figure 11. Experiments were conducted on a 10X10 grid environment with two terminal states, with values 10 and 100 respectively

using our modified SARSA(λ) algorithm (illustrated in Figure 10) with an ϵ -greedy policy. We attenuate ϵ according to the schedule illustrated in Figure 12.b.

5.3.3 Preliminary Analysis

Figure 13 illustrates the progression of the confidence metric for the best (most valuable) action in each state over the course of the episodic experiments. Areas represented in white are those for which the agent's computed standard error falls beneath the required threshold, indicating a strong degree of certainty about the stability of the value. Areas in black represent the values that fall above the required threshold. The first attribute to note is the anomaly that occurs in the graphic for the 100th episode. The agent appears to be confident in the values for the state-actions in the lower left corner, which occur in close proximity to the start state (1,1). In reality,

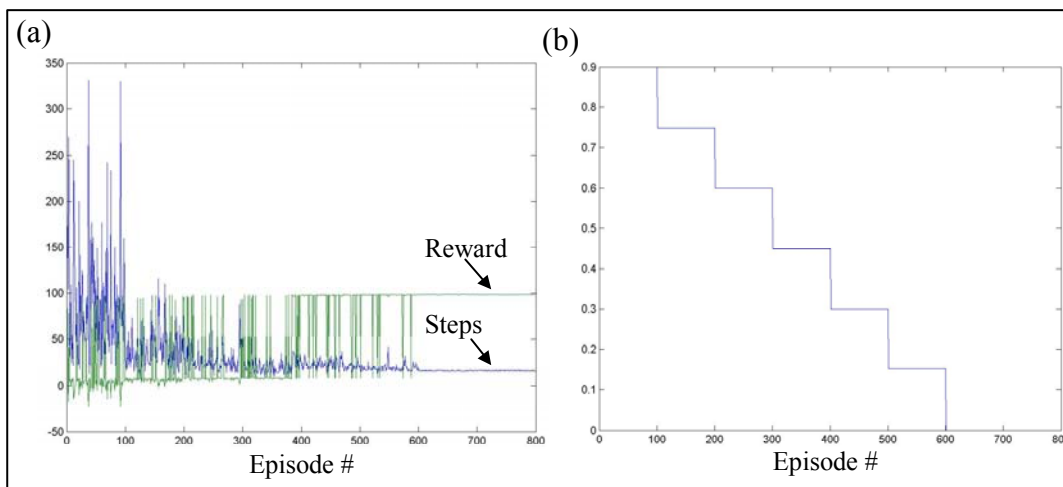


Figure 12. (a) Convergence of reward and # of steps executed by agent. (b) Attenuation of ϵ value.

at the 100th episode those values are still far from the true value associated with the optimal policy. This anomaly occurs due to the nature of reinforcement learning, since the values for state-actions that are far from the reward states receive very little propagated reward during the early stages of learning.

This problem is further exacerbated by the fact that our experiment has a consistent start state for each episode, meaning that the states and actions in the lower left quadrant are visited with a high frequency. These two situations combined fool the agent into believing that the values for these state-action pairs are stable. We have begun to experiment with potential remedies as our research continues. What begins to emerge in the subsequent measurements is a band of certainty around the optimal policy. Terminal states are denoted as uncertain because no actions are executed within them and consequently no measurements are made. Areas denoted as uncertain remain so because the agent does not visit these states often enough to acquire enough stable data for the standard error to fall below the threshold. It is unclear at this point of our analysis whether the Q-values fail to converge or the agent just doesn't experience the state-

actions for a long enough period of time after convergence. The agent's ability to measure a stable value lags behind the actual appearance of the stable value because the agent needs to see a sufficiently long sequence of values to compute the measurement.

Figure 14 illustrates further measures for sub-optimal actions, indicating a relatively large set of information for which the agent has achieved a desired level of certainty.

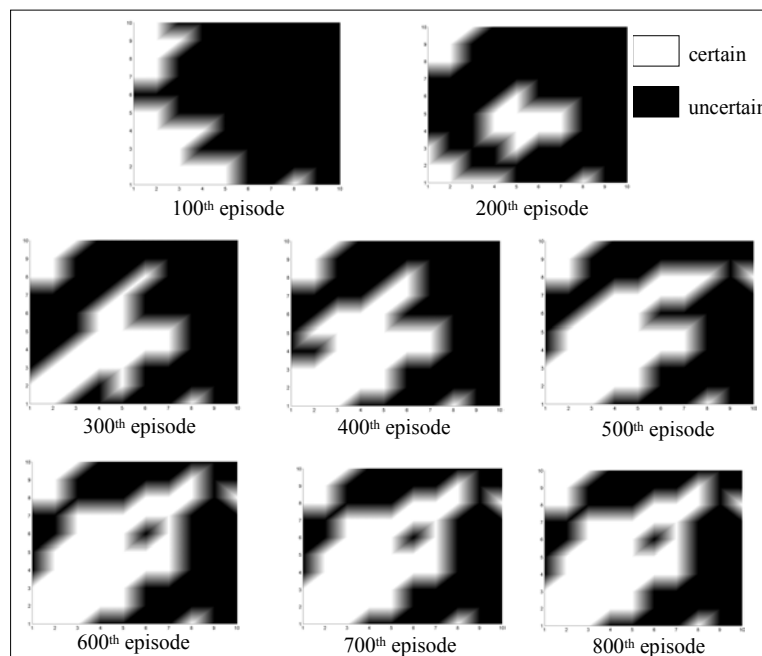


Figure 13. Certainty measurements for optimal actions, at 100-episode sampling increments

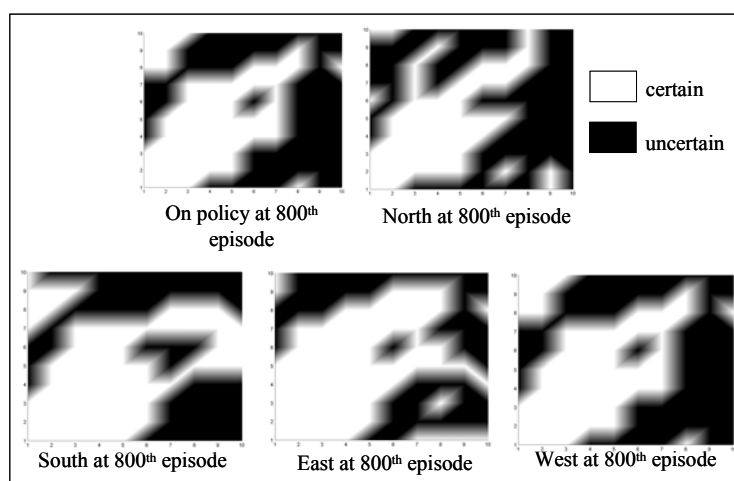


Figure 14. Measures for all actions at 800th episode.

5.4 Continuing Research and Future Direction

Now that we have developed a theoretical premise for measuring the quality of an agent's knowledge, we have begun to develop a full-scale control algorithm for individual agents and a means by which this algorithm can incorporate notions of multi-agency and the added dynamics inherent in a Multi-Agent System. The initial objective is to develop a *Partially Observable Markov Decision Problem* (PO-MDP) [1, 7] formulation, which will allow agents to minimally interact with peer agents and coherently execute search and surveillance actions.

Since our work is predicated on the notion that agents are making sequential decisions, it is important to formally describe this problem family and discuss how our research will contribute to current methods.

5.4.1 Sequential Decision Problems

Sequential decision problems are generally discussed in the context of two discriminating features. The first is the Markov property, which requires that the agent's percept provide enough information to identify the current state of its environment at each decision point. The second is the accessibility of the environment, which describes the degree to which the agent can directly observe the true state of its environment. A fully accessible environment is one in which the agent's percepts are direct observations of the full set of state variables. Markov Decision Problems are those that obey the Markov property in fully accessible environments. Formally, an MDP can be specified as a 4-tuple $\{S, A, T, R\}$, where:

- S is a finite set of environment states
- A is a finite set of actions available to the agent
- R is a reward function that maps from the states S to real-valued rewards $R: S \times A \rightarrow \mathcal{R}$
- T is a transition function that maps $T: S \times A \rightarrow \Theta(S)$, where $\Theta(S)$ is the state transition function specifying a discrete probability distribution $P_{ss'}^a$, which is the probability that the agent executing action a in state s will transition to state s' for all $a \in A$ and all $s, s' \in S$

More generally, we refer to a k th-order MDP, where the agent must consider the last k states visited to compute the transition probabilities as $P(S_n = s_n | a_{n-1}, S_{n-1}, \dots, S_{n-k})$. By allowing $k \rightarrow \infty$, any fully accessible sequential decision problem can be formally represented as an MDP, although it would not always be practical to solve it as such. The definition above refers to a 1st-order MDP.

Unfortunately, like the TASK family of problems, many realistic problem environments are not accessible [4, 6] and therefore policies cannot directly be derived using standard MDP methods. These problems are instead considered Partially Observable Markov Decision Processes (POMDP), a sequential decision problem in which the agent must select actions without certain knowledge of the current state of the environment (inaccessible). The general solution approach for a POMDP is to compute a probability distribution over the possible states at each timestep and treat the problem as an MDP [1, 6], using the distribution as the state signal. Formally, a POMDP can be represented as a 6-tuple $\{S, A, T, R, O, B\}$ where:

- S, A, T, and R are defined as they are for the MDP
- O is a finite set of observations that the agent receives at each timestep in place of the true state signal
- B is a belief model that maps $B: S \times A \rightarrow \Theta(O)$, where B_{os}^a is the probability of observing o in state s after the agent has executed action a .

The agent would then use the probability distribution across states to make a decision about which action to execute.

5.4.2 Problem Parameters

We will abstract away much of the MAS complexity for the initial problem formulation, allowing us to test initial concepts and put a mechanism in place on which we can build more complex approaches to more complex problems.

We would like our agent controller to eventually be capable of dealing with a problem like the following:

An agent possesses information about 8 tasks of the agent's type, 3 known, varied-value tasks that are distributed geographically and 5 unknown tasks, each with different ages (quality) associated with the knowledge, and 3 agents within detection (but not communication) range, and 1 agent within communication range. At the agent's disposal is a set of actions that will allow it to act on its current knowledge or seek out additional knowledge. The agent can attempt to service the known tasks, to communicate with known peers, or explore the environment for new tasks or peers. What should the agent do?

The primary question is, under what set of circumstances is each action a provably good choice?

This problem is described from the perspective of the agent's knowledge. Notice that there are a few salient pieces of information missing that make this problem very difficult, namely the true location, type, and value of all unserved tasks, the locations and types of all agents, and the intentions of all agents. And of course one final complicating detail is that all of this information changes at each time step because of the unobservable randomness associated with the actions of other agents and the evolving system dynamics. Addressing this problem in its full complexity is a goal toward which we will work, but not the starting point.

We're starting our work from the basic formulation of a Partially Observable Markov Decision Process (PO-MDP). Solving a PO-MDP means providing the agent with a means for selecting a good action at each decision point, based on the cost of that action in a particular state²⁰. Since a PO-MDP assumes the true state is unavailable, the agent will make the calculation based on a probability distribution over states. We compute this probability distribution based on a model of state transitions and an estimation function that probabilistically maps observed evidence to states. The cost is then computed via a function that yields the cost of actions based on the

²⁰ This isn't to imply that the policy involves only a 1-step measurement of value/cost. The true policy may rely on a cost function that includes a notion of looking ahead across some finite horizon of transitions reachable from the current state, complicating the cost calculation. Recursive definitions of cost/value might incorporate this notion

probability distribution over the current state and the probability of the states to which the agent could transition. These elements are detailed further in subsequent sections.

In our initial problem formulation, the general goal of any agent is to accumulate the most reward for servicing tasks. There are numerous circumstances one could create under which complex strategies, such as short-term sacrifices for long-term benefit or sacrifices for the benefit of others, could be motivated. This is not the focus of our initial work. At the heart of our problem is the notion of the uncertainty of information obtained through communication with

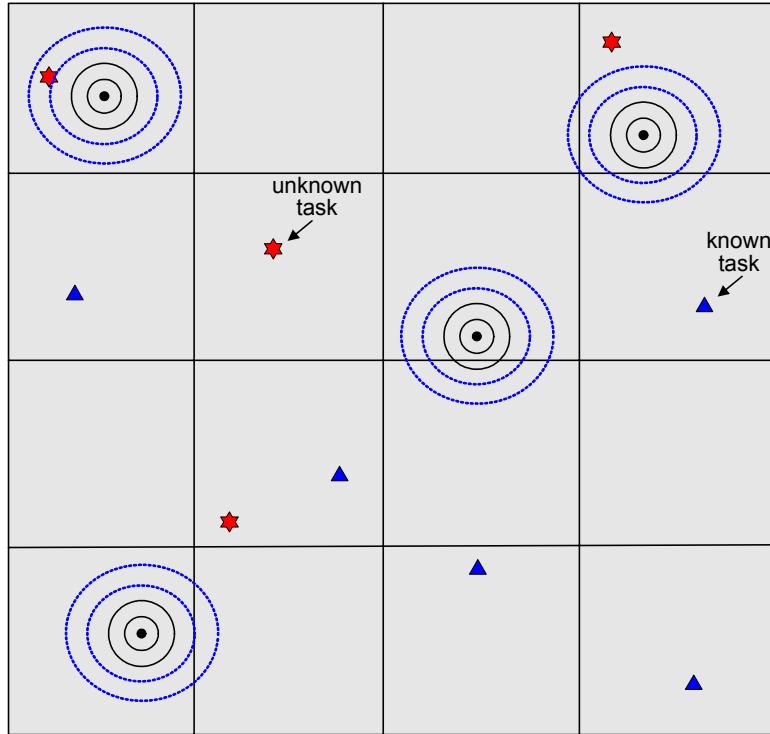


Figure 15. Agents navigating through a physical space with known and unknown tasks. Agents have distinct task surveillance, task detection, agent communication, and agent detection ranges, respectively illustrated by the four rings surrounding the agent, starting from the innermost ring

peer agents or from the age of globally broadcasted knowledge. Risk is associated with the age of information in the form of the cost of servicing a task. Cost may simply be a function of time, in that there is a fixed penalty associated with each time step that can only be offset by accumulating reward from servicing tasks. Time misspent pursuing tasks that no longer need servicing is an example of the cost associated with the risk of uncertain information. Assuming the agents travel at a finite speed and all task values and information ages being equal, closer tasks are more desirable. Since all task values and information ages are not equal, the agent then must estimate the utility of attempting to service a task, based on a belief that the task remains unserviced and the anticipated value of servicing the task. Alternate actions, such as searching or communicating, may be more beneficial, depending on the circumstances.

Figure 15 illustrates the basic problem setup. Agents are navigating through the environment, searching for tasks and attempting to survey them if found. Searching for a task only identifies the existence of a generic task within a field of view (FOV). The surveillance detector subsequently receives type, value, and location information. Agents also encounter their peers, with whom they can then exchange collected observations. The environment is comprised of a set of tasks that are *known* and *unknown*, of which the definitions are still somewhat flexible. For now, we assume that when the experiment starts, agents are provided with the locations, types, and values of all known tasks but only the types of all unknown tasks. Tasks that are serviced are replaced at a constant rate and information about those tasks is centrally broadcast to all agents, either the full location, type, and value for known tasks or just the type for unknown tasks. Serviced tasks remain detectable until they are replaced. The agent receives no value for surveying a task that has already been serviced.

Despite the global broadcast of data, this information is still uncertain due to the limited observational perspective of the agent, combined with the presence of other agents in the environment. Once the task has appeared in the environment, regardless of a known or unknown location, the agent does not know the task's fate until it either confirms for itself or is told by another agent that it has been serviced. In other words, pursuing a known task is not a sure thing.

5.4.3 PO-MDP Controller Formulation

State Transition Model – Using control theory terminology, the state of the system at time $k+1$, x_{k+1} , is determined by the function $f_k(x_k, u_k, w_k)$, where u_k is the control executed by the agent in the previous time period and w_k represents a random system disturbance with a known probability distribution. One might assume that in the TASK setting, the random disturbance would be characterized by the randomness attributed to the actions of other agents and the addition of tasks to the environment by the central supervisor. We will need to empirically determine the state transition model, $P(x_{k+1} | x_k, u_k, w_k)$, which will provide the agent with the distribution over the outcomes of the function f_k .

Estimation Function – Again, in control theory terminology, the evidence observations are denoted at time k by z_k , determined by the function $h_k(x_k, u_{k-1}, v_k)$, where x_k is the current state, u_{k-1} is the previous control, and v_k a random observation disturbance. We will need to empirically determine the probability distribution for the set of possible observations, given by $P(z_k | x_k, u_{k-1}, v_k)$, which will provide an essential component necessary for the agent to compute the probability distribution over states. Note, this does not take into account other mechanisms, like recursive estimation. Many different possibilities abound.

Cost Function – Lastly, a cost function uses the computed distribution over states, with known costs of state transitions, to compute expected costs for each possible control.

Henceforth the probability distribution over true states will be referred to as the *belief state*. We are not currently addressing online adaptation of model or policy parameters and we will assume for now that our policy function is a relatively simple function of cost.

Figure 16 illustrates a basic formulation for the TASK agent as a PO-MDP controller. The subscript k represents the current time step. The following is a set of term definitions:

x_k – the true state of the system at the current time step,

u_k – control executed by agent,

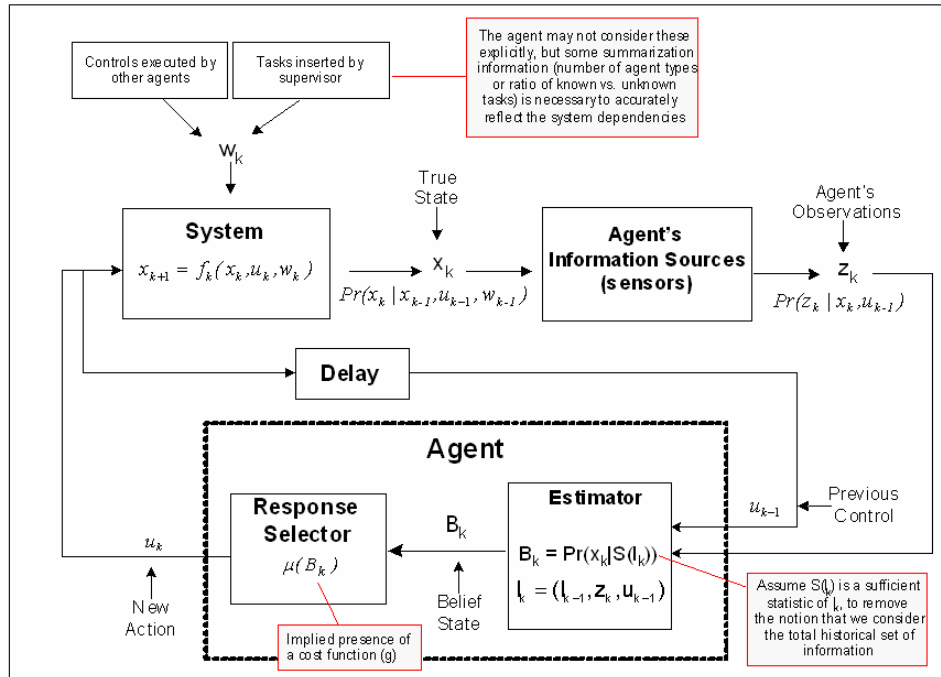


Figure 16. A PO-MDP controller formulation of a TASK Agent.

z_k - set of current observations available to the agent,

I_k - the total set of information used for computing belief state, including current and historical observations and previous actions,

$S(I_k)$ – a sufficient statistic representing the minimum information from I_k needed to compute the belief state,

B_k - the agent's estimated probability distribution over the set of true states (belief state),

g – a cost function yielding the cost of each action in each state,

μ - a policy function that selects the next action based on the belief state and the cost function

System States

As part of our PO-MDP formulation, we must define the set of states that the system can assume and the possible transitions between those states. Figure 17 illustrates our initial approach to the

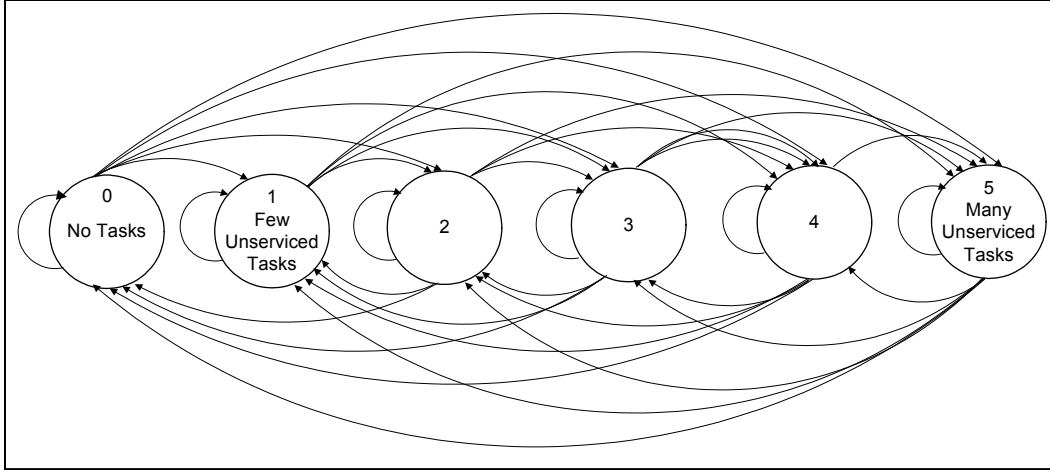


Figure 17. State transitions reflecting discrimination on the task density within the agent's area of interest

system states. This delineation is based on the notion that an agent is only interested in things that are happening within a finite Area of Interest (AOI), which is much smaller than the entirety of the physical space within which the experiment takes place. So, the states denoted in Figure 17 indicate the true density of tasks within the area of interest. The system can clearly transition between any two states, based on the influence of other agents in the environment or the addition of tasks by the simulator.

The agent will use its sensors and interaction with peer agents to collect observations, with which it can probabilistically compute the likelihood that the system is in any particular state, and then select an appropriate action.

5.4.4 Observations

First, it is important to note that in our problem formulation there is a difference between the large set of information that the agent acquires and stores and the smaller set of information the agent considers as observations in the control theory sense of the word. The agent maintains information about every cell in the environment, illustrated in Figure 18. This is the information as it was collected by or reported to the agent, but may not be accurate due to the passage of time. The agent uses the information from the area of interest for its own decision-making, but shares all information with peers. A set of information is then filtered from the total map into a set of observables relevant to the agent's area of interest, with which the agent then computes its belief state.

Table 3 lists the source and format of the information the agent may receive over its lifetime. From this a set of observables will be distilled for the computation at each agent decision point.

Table 3. Source and content of all raw information, from which the agent constructs a set of observations.

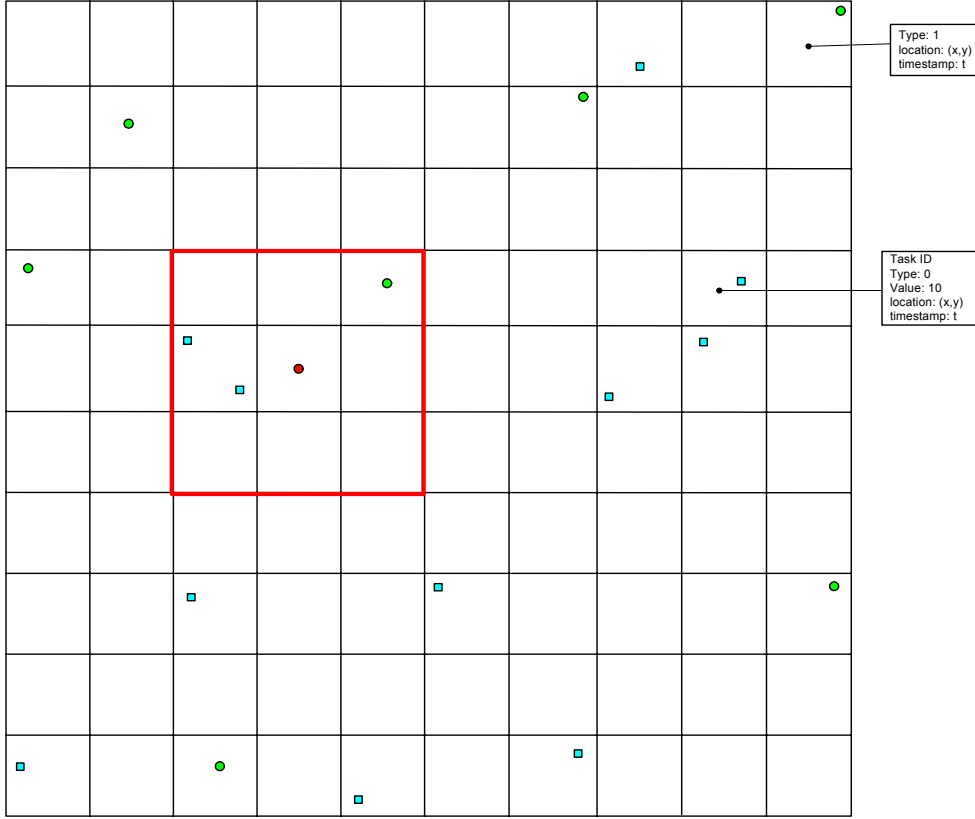


Figure 18. Agents maintain information about the entire environment, but limit decision making to include information about a limited area of interest

Source	Description	Representation	Notes
Simulator	Known Tasks	$T = [\{p_1, v_1, t_1, s_1\}, \{p_2, v_2, t_2, s_2\}, \dots, \{p_n, v_n, t_n, s_n\}]$	Where p_i is the task's location coordinate in Cartesian space, v_i is the value of the task, t_i is the type of the task drawn from the set of types $t_i \in \{0, 1\}$, and s_i is the time step at which the information was originally acquired from central broadcast.
	Unknown Tasks	Maybe just the number of unknown tasks at system start and incremental updates to indicate the addition of a new unknown task	
Search Detector	Unknown Tasks	$T = [\{p_1, s_1\}, \{p_2, s_2\}, \dots, \{p_n, s_n\}]$	Where p_i is the location coordinate in Cartesian space representing the center of the circle within which the task resides and s_i is the time

			step at which the task was detected.
Surveillance Detector	Known/unknown Tasks	$T = [\{p_1, v_1, t_1, s_1\}, \{p_2, v_2, t_2, s_2\}, \dots, \{p_n, v_n, t_n, s_n\}]$	Where p_i is the task's location coordinate in Cartesian space, v_i is the value of the task, t_i is the type of the task drawn from the set of types $t_i \in \{0, 1\}$, and s_i is the time step at which the task was detected.
Peer Agents	Known/unknown Tasks	$T = [\{p_1, v_1, t_1, s_1\}, \{p_2, v_2, t_2, s_2\}, \dots, \{p_n, v_n, t_n, s_n\}]$	Where p_i is the task's location coordinate in Cartesian space, v_i is the value ²¹ of the task, t_i is the type of the task drawn from the set of types $t_i \in \{0, 1\}$, and s_i is the time step at which the information was originally acquired by the first agent.
Agent Detector	Peer Agents	$A = [\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_n, t_n\}]$	Where p_i is the agent's location coordinate in Cartesian space and t_i is the type of the agent drawn from the set of types $t_i \in \{0, 1\}$.

Computing the Belief State:

Part of our research work is to devise a method by which the agent can use observed information to compute a belief state (B_k), which is a probability distribution over the set of possible states. The following is a discussion of our initial approach to computing the belief state. We introduce the parameters of the function, but the actual function will be determined through empirical analysis.

Assumptions:

- 1) All tasks are the same value
- 2) The total number of tasks is known
- 3) The distribution of task types is known (e.g. 20% type A, 30% type B, etc)
- 4) The total number of agents is known
- 5) The distribution of agent types is known

The agent should use its observations to compute its belief about the density of tasks within the area of interest. The function, h , which the agent uses to compute the probability over states, should include the following as parameters:

$$B_k = h(\{(t_1, s_1), (t_2, s_2), \dots, (t_n, s_n)\}, k(x), u(y), a(z)) -$$

²¹ Receiving a task value of zero from a peer agent indicates the task has already been serviced

- The set of known (from simulator and peers) tasks in the AOI, represented as pairs of task IDs (t_i) and time stamps (s_i)
- $a(z)$, the number of agents in the area of interest, with the parameter z being the total number of agents in the environment. Initially, we might be able to achieve this by assuming a uniform distribution of agents (pursuing uniformly distributed tasks) and then just calculating the number of agents expected to be in the physical space of the AOI. More advanced methods might include using observed or reported information about agents and trajectories.
- $u(y)$, the number of unknown tasks expected to be in the AOI. Again, we can probably use the expected value, given the uniform distribution and number of unknown tasks believed to be remaining.
- $k(x)$, the number of known tasks expected to be in the AOI. Again, we can probably use the expected value, given the uniform distribution and number of known tasks believed to be remaining.

Example:

- 1) The decision-making agent is of type A
- 2) Total type A tasks: 50; 45 known, 5 unknown;
- 3) Number of other type A agents: 10
- 4) Proportionate size of AOI: 20%
- 5) 9 type A known tasks were originally reported in the AOI (following the uniform distribution), 2 have since been reported as serviced, leaving 7 for which no new information has been received:
- 6) Since we have a total of 5 unknown tasks and our AOI is 20% of the total physical space, we expect to find 1 unknown task within the AOI.
- 7) So we start the calculation with the notion of 8 potential tasks in the AOI. We're then left to determine the impact of the age of the information about each task and the density of type A agents.

With this information we compute a distribution over the states listed above.

5.4.5 Actions

The following actions are available to the agent at each decision step:

Wide Area Search (WAS) –WAS causes the agent to execute a search pattern over a large area of the physical space while applying the Search Detector.

Fine-Grained Search (FGS) –FGS causes the agent to execute a search pattern over a smaller area of the physical space while applying the Search Detector. This small area may be centered around the known location of a task or the center of the area in which a task has been detected.

Service (SVC) – SVC causes the agent to apply the Surveillance Detector over a small area of the physical space in which a task has been detected.

Communicate(a_i) –Communication is reciprocal, so an agent both sends and receives information. All information is exchanged, more than just the set of observables for the AOI.

Move(p) – Moves the agent to point p in the physical space. The point is 2D (Cartesian) in our current configuration.

5.4.6 Policy

Typically, the policy is just a mathematical function of utility of a state and the probability that an action will transition the agent to that state, look-aheads, etc not withstanding. We assume the probability distribution is handled by the estimator, so the explicit difficulty here is the utility function, otherwise known as the cost function. The nature and effect of agent interaction may make the cost function difficult to capture empirically.

Interaction between agents in a Multi-Agent System typically has an explicit connotation, such as negotiating joint actions or directly observing and reasoning about another agent in order to select complimentary actions. Our initial problem formulation is scoped to include only exchange of observation information (e.g. task locations/values/types), with the possible exception of explicitly considering histories of other agents to appropriately weigh search strategies or to weigh the value of communication with a particular agent.

A significant question is then how to represent the expected value/cost of exchanging information and how to represent the resultant state transition. Communicating information does not change the external state of the environment, but rather changes the agent's state of knowledge. Aside from the question of how this is modeled, how is this transition valued? Is the measure quantitative, in terms of the amount of information received or is it qualitative, in terms of the impact on the agent's ability to make good decisions and reap the subsequent reward? Are there implicit costs associated with communicating, such as the delay in direct action? E.g. time spent communicating is time not spent searching or servicing. Is there a way of measuring the cost associated with poor action choices based on bad (old) information, received from another agent? Or, can all of this be avoided by simply making search and communication default actions, chosen uniformly when no task is available to attempt to service? These are questions we will endeavor to answer as our research progresses.

5.5 Conclusions and Continuing Research

We have successfully conducted research to develop statistical methods by which agents operating in a Multi-Agent System can efficiently share knowledge. Our current work is designed to provide control logic and supporting knowledge structures for single agents

operating in cooperative Multi-Agent Systems, within which we can further explore the value of cooperative learning and knowledge sharing.

To facilitate this formulation of a PO-MDP controller, we will conduct experiments to develop the correct level of granularity for representations of system state and agent observations, which will allow agents to tractably compute probabilistic beliefs about the conditions of the dynamic environment. Once we have developed an appropriate agent mechanism for assessing the state of the environment, we will turn our attention to the design of policy functions, which include methods for determining the cost and efficacy of agent actions. Combining the state models with the policy functions will provide the agent with a mathematical mechanism for decision-making in the uncertain Multi-Agent System environment.

Our near-term work will be focused on completing these objectives and conducting initial experiments to determine the applicability of a PO-MDP agent controller within a Multi-Agent System environment. We expect our initial design to undergo iterative refinement during this process. Once we have arrived at a stable design for the agent controller we can then design and evaluate our formal methods for cooperative learning and knowledge sharing.

6 Project Conclusions and Continuing Work

6.1 Open Experimentation Framework

ALPHATECH has accomplished the following major OEF milestones:

- Successfully defined a MAS UAV problem, administered the CAHDE REF, and coordinated the participation and research of multiple TASK project PIs.
- Identified and refined seven major critical elements for MAS research and evaluation and defined their dimensions
- Developed detailed OEF problem specifications and metrics for Adaptation and Coordination critical elements, which will provide PI researchers with a common framework for evaluating their solutions
- Defined and specified baseline parameters for the UAV-S(1) surveillance problem, which includes the configurable dimensions of the UAV problem and measurement criteria. This provides a common environment for PI research and evaluation.
- Engaged all project PIs on OEF definition, problem specification, and integration of their research

ALPHATECH's future OEF work will entail the following:

- Continue specifying the OEF in terms of the major MAS design critical elements so that MAS design dimensions are well understood and measurement criteria can be developed
- Fully define the UAV-S(2)-(5) problems to facilitate research along new dimensions and additional levels of problem complexity

- Supporting the PIs to integrate their research into the OEF so that a common evaluation framework can be developed
- Supporting the DARPA Program Manager with reports and status
- Continue to develop and maintain the problem generator to support the UAV problem specifications

6.2 Testbed for Taskable Agent Systems

ALPHATECH has accomplished the following major testbed milestones.:

- Developed a multi-threaded software system capable of supporting:
 - Well defined APIs supporting multiple interactive agents with different underlying implementations so that disparate agent technologies can be evaluated within the same framework
 - Fully 3-dimensional environment to support realistic problem formulations
 - Six degrees of freedom for UAV motion to support more complex and realistic agent decisions
 - Relational database for extensive experiment data logging to support experiment analysis
 - XML configuration file to adjust the many system parameters and to support repeatable experiments

ALPHATECH's future testbed work will be focused on adding the following features:

- Support for moving targets
- Providing facilities for unreliable information to support research addressing uncertainty and trust
- Supporting cross mission tasking scenarios to support research requiring agents to negotiate joint actions
- Adding time critical targeting to support research approaches that can deal with computational bounds on decision algorithms
- Including terrain and geographic features and the effects of terrain on line-of-sight sensing and communication

6.3 Multi-Agent System Research

ALPHATECH has accomplished the following major research milestones.

- Conducted extensive research efforts focused on the development of new mechanisms for coordination and interaction of agents operating in a Multi-Agent System.
- Developed a mechanism for measuring quality of knowledge learned through Reinforcement Learning so that agents can decide what warrants knowledge exchange
- Conducted experiments to evaluate quality measurement methodology

- Began development of a Partially Observable Markov Decision Process framework for an agent controller capable of operating within a multi-agent system environment. Will support agent decision-making and actions and provide a foundation for cooperative learning research

ALPHATECH's future research work will be focused on:

- Completing the design of the PO-MDP agent controller so that the environment states and agent observations are fully specified and mathematically consistent. This provides the foundation for reasoning and action.
- Implementing the agent controller within the TTAS to conduct robust UAV experiments with baseline UAV-S(1) problem
- Conducting further experimentation and research with our cooperative learning algorithms to allow agents to exchange and benefit from the learned knowledge of other peer agents, dramatically reducing the computational complexity of the online learning problem and increasing the efficacy of the agents

7 References

- 1 Bertsekas, D. P. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2000.
- 2 Bury, K. *Statistical Distributions in Engineering*. Cambridge University Press, Cambridge, UK, 1999.
- 3 Kaelbling, L.P., Littman, M.L., and Moore, A.W. *Reinforcement learning: A survey*. Journal of Artificial Intelligence Research, 4:237--285, 1996
- 4 Littman, M. L., Cassandra A. R., Kaelbling, L. P. *Learning policies for partially observable environments: Scaling up*. Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann publishers Inc.: San Mateo, CA, 1995.
- 5 Milton, J. S. and Arnold, J. C. *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*. McGraw-Hill, New York, NY, 1990.
- 6 Parr, R. and Russell, S. *Approximating Optimal Policies for Partially Observable Stochastic Domains*. In Proc. Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada, 1995.
- 7 Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall Inc, 1995.
- 8 Sutton, R. S. and Barto, A. G. *Reinforcement Learning. An Introduction*. Cambridge, MA: MIT Press, 1998