AFRL-IF-RS-TR-2003-61
Final Technical Report
March 2003

TEAM LEADER: AN APPROACH TO MIXED-INITIATIVE AGENT TEAM MANAGEMENT AND EVALUATION

BBNT Solutions LLC

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

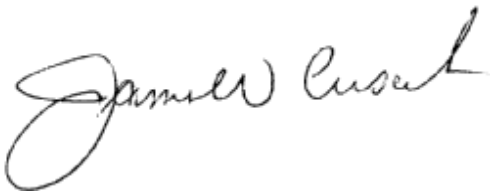
AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

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-2003-61 has been reviewed and is approved for publication.

APPROVED: 

JOSEPH A. CAROLI
Project Engineer

FOR THE DIRECTOR: 

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2003	Final Jun 98 – Nov 02

4. TITLE AND SUBTITLE
TEAM LEADER: AN APPROACH TO MIXED-INITIATIVE AGENT TEAM MANAGEMENT AND EVALUATION

5. FUNDING NUMBERS
C - F30602-98-C-0168
PE - 63760E
PR - AGEN
TA - T0
WU - 11

6. AUTHOR(S)
Mark Burstein, David Diller, Alice Mulvehill, Brett Benyo, and Ed Pattison-Gordon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBNT Solutions LLC
10 Moulton Street
Cambridge Massachusetts 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFSF
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-61

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Joseph A. Caroli/IFSF/(315)330-4205/ Joseph.Caroli@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The rapid growth in research and development of agent-based software systems has led to concerns about how human users will control the activities of teams of agents that must actively collaborate. Practical multi-agent systems will often be comprised of small teams of heterogeneous agents, under direct supervision by users acting as "team leaders". This research focused on a number of areas critical to tasking and managing teams of humans and agents, including techniques for 'agentizing' legacy systems, Human-Computer Interaction techniques for interactively defining tasks and roles and communications between agents, mechanisms for interactively forming and controlling teams of agents as well as summarizing and visualizing agent task status, methods for translating information across different ontologies, and mechanisms for monitoring and controlling agent communications based on semantic content. A key contribution of this work has been the development of an approach to dynamic information sharing among agents on teams, alleviating human managers from the need to coordinate information awareness. The work was demonstrated in a number of large-scale demonstrations including the Coalition Agents eXperiment (CoAX) and the Mixed-Initiative Agent Team Administration (MIATA) demonstration.

14. SUBJECT TERMS
Agent Based Systems, Software Agents, DARPA Agent Markup Language, DAML, Software Agent Demonstration, Agent Teams, Agent Information Sharing

15. NUMBER OF PAGES
44

16. PRICE CODE

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

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

1. INTRODUCTION

The DARPA Control of Agent-Based Systems (CoABS) Program has ushered in an era of fast-paced growth in the development of agent-based systems technologies. The functionalities, capabilities, and limitations of agents have evolved rapidly in recent years. Through our efforts in the CoABS program we have begun to understand what the capabilities of software agents can be, and we have a better understanding of how to make effective use of large numbers of agents in the accomplishment of specific, large scale tasks. Specifically, we have focused on strategies and mechanisms for making effective use of heterogeneous agents in tasks requiring some degree of inter-agent or human-agent coordination. It has been our belief throughout this program that software agents are best utilized as parts of carefully composed teams of software (and human) agents where humans act as the primary team leaders or managers, directing the team's behavior and managing the team's activities during execution, especially when problems occur.

However, many of the distributed software components that are needed in real-world systems are not agents at all by most current definitions. Our approach has been that we must expect that a range of "agent" and "less than agent" software components will be involved, and we have developed mechanisms to characterize their functional capabilities, domain knowledge, information requirements, and decision authority, in order to properly task them. Furthermore, the ability to 'agentize' legacy systems in order to improve interoperability has been an important issue, and one in which we have developed techniques and experience.

Generally, our approach has been to simultaneously develop mixed-initiative user support agents and tools and an environment in which we can experimentally explore approaches to user interaction with these team building and managing agents. We believe that effective techniques and tools for humans to use in managing and organizing agent teams must be derived experimentally. To this end, we developed a testbed using the OMAR (Operator Model Architecture) agent simulation environment (Deutsch, 1998; Deutsch & Adams, 1995): a sophisticated environment for discrete-event simulation modeling of goal-directed agent behavior developed at BBN, for experiments in human-centered agent team design and management.

This testbed consists of an agent simulation environment that enables us to explore techniques for tasking agents using domain objectives and terms. The testbed has also allowed users to organize teams and address coordination issues, such as the most effective use of mediators, and the selection of effective organizational control strategies for different classes of tasks. By using OMAR as a simulation tool supporting agents with explicit goal and process representations we have been able to model the wide variety of software agents that have been necessary to explore the issues addressed herein.

In order to formally evaluate our findings, we participated in a series of three Technology Integration Experiments (TIEs). These TIEs have involved a series of demonstrations of mixed-initiative agent-based computing in the context of large military organizations.

The NEO (Noncombatant Evacuation Operation) TIE illustrated the use of agent technology for cooperatively planning and executing a hypothetical evacuation of US civilians from a Middle Eastern city in response to an escalating terrorism crisis. The NEO TIE involved the coordination of a number of agent systems whose agents were used to evaluate a crisis situation, form an evacuation plan, follow an evolving context, monitor activity, and dynamically re-plan. The NEO TIE demonstrated the interoperability and use of multiple disparate agent systems for aiding humans in effectively monitoring the scenario, retrieving and fusing information for immediate use, and planning and re-planning an emergency evacuation.

The MIATA (Mixed-Initiative Agent Team Administration) TIE involved a series of demonstrations of mixed-initiative agent-based computing as part of a disaster relief effort, modeled on the events surrounding Hurricane Mitch in 1998. The goal was to explore and validate the potential for the CoABS

Grid to support disparate agent-based systems functioning effectively in collaboration with humans in a dynamic, distributed organization.

Most recently, the Coalition Agents Experiment (CoAX) TIE was created to provide a rich and militarily relevant setting for experimenting with agent-based systems for coalition operations. One of the critical concerns in any coalition operation (military or civilian) involves protection of sensitive or proprietary information. The CoAX TIE models the complexities and nuances of the relationships between different countries that make up the coalition peacekeeping force, including varying levels of trust, misinformation and deception, and the need for varying degrees of dynamic information sharing.

2. OBJECTIVES

It has been our contention that the planning of effective agent teams is no less complex than the planning for effective human teams. In different ways, it is more complex as the capabilities and limitations of software agents are more pronounced. Software agents will typically not possess human-like intelligence and common sense. Most are more like "idiot-savants" in their dedication to a single purpose. As such, plans to coordinate their behavior need to be more precise. Issues of team member selection and coordination, and proper communications content in exchanges between agents need to be carefully planned. Team performance needs to be monitored for "disconnects" and failures anticipated or discovered as soon as possible during execution. Our claim is that effective support for human managers of agent teams is and will be of paramount concern for the foreseeable future, and that the agents must become more adaptive and reflective to enable humans, even assisted by planning agents, to manage them effectively.

This support for humans requires a number of areas be addressed. We have focused our research on a number of areas critical to tasking and managing teams of humans and agents, including:

1. Techniques for 'agentizing' existing systems to enable them to be more easily integrated into the functions of larger, human managed organizations.
2. The development of an environment in which we can experimentally explore approaches to human and agent interactions.
3. Human Computer Interaction techniques for interactively defining tasks and specifying roles and communications between agents.
4. Mixed-initiative support agents to help users identify appropriate agents for tasks, and ascertain that agents can communicate compatibly, satisfy timing dependencies, and provide structural support for execution status monitoring.
5. Mechanisms for interactively forming and controlling teams of agents in support of human needs and objectives.
6. Support for the collection, summarization and visualization of agent task status, to convey to users the impact of problems on major team objectives.
7. Support for dynamic retasking to surmount execution-time problems, changing conditions and objectives.
8. A model for the establishment of cooperative information sharing among teams of agents.
9. Techniques for automatically translating information across differing ontologies thereby allowing agents to 'speak the same language'.
10. Exploring the use of DAML-S as a language to express the capabilities and services of agents.
11. Techniques for monitoring and controlling agent communication based on the semantic content of the communication messages, which enable agents to communicate interoperability.

3. NONCOMBATANT EVACUATION OPERATION TECHNOLOGY INTEGRATION EXPERIMENT

The NEO TIE was developed to illustrate agent technologies for collaboration, information retrieval and transformation across distributed data sources, and for agent supported scheduling and planning. This was done in the context of a hypothetical scenario in which US civilians must be evacuated from Kuwait City in response to a bomb explosion at the International Conference on Petroleum and the Environment. In the NEO TIE news and weather information were retrieved from a number of different sources and used as part of an initial planning phase. Primary and contingency flight plans were devised using both commercial and military airlifts. Evacuees were located using a number of different information sources, and evacuation routes planned. Evacuation missions were simulated and rehearsed, with various dynamic events, requiring active scene monitoring and dynamic replanning. The NEO TIE demonstrated the interoperability and use of multiple disparate agent systems for aiding humans to effectively monitor the scenario, retrieve and fuse information for immediate use, and to plan and re-plan an emergency evacuation.

Our contribution to this TIE was an agent-wrapped version of the Camps Mission Planning System (MPS) that was under joint development by BBN and Kestrel Institute for the Air Mobility Command. As the main scheduling engine part of MPS was written in LISP, we developed a general-purpose agent wrapping mechanism to enable MPS to talk to other software agents that were using the Retsina implementation of the KQML agent intercommunications language. This tool, called the LISP-KQML Proxy, was subsequently used by several other research teams within the CoABS program, including the University of Michigan and CMU, and a revised version is now available for use with the CoABS GRID.

3.1 AGENTIZING THE CONSOLIDATED AIR MOBILITY PLANNING SYSTEM (CAMPS) – MISSION PLANNER (MP)

BBN developed the 'agentized' version of the CAMPS Mission Planner (MP) for experimental use in mixed-initiative multi-agent systems that included logistics elements. The CAMPS Mission Planner (Burstein & Emerson, 1999) is a scheduling system for cargo aircraft developed jointly by BBN and Kestrel Institute for the Air Force. CAMPS-MP takes as inputs a set of 'requirements', each consisting of quantities of cargo and people to be moved from one airport to another during some time interval. It produces detailed schedules specifying the times at which an aircraft of some type will fly from where they are based to pick up cargo and carry it to its destination, and then return to home base. Requirement sets can be quite large, numbering hundreds or even thousands of elements, and hundreds of tons of cargo, from tens of locations. Numerous constraints must be satisfied simultaneously to produce valid schedules, which the scheduler can do in seconds or minutes.

A simplistic view of the task faced by a user of the Mission Planner is, given a set of requirements, to specify a set of suitable aircraft resources, the ports to be made available for refueling (or locations for aerial refueling), should that be necessary, and to ensure that the schedule produced moves all of the requirements by their due dates. The scheduler will fail to schedule flights for all of the cargo requirements if there are constraint violations, or insufficient resources provided, in which case some cargo may be scheduled to arrive late or not at all. As originally developed, the CAMPS Mission Planner had a traditional graphical user interface for specifying input parameters, and a variety of views of the product schedules produced, including maps, tables and GANTT charts. All user interactions were by keyboard and mouse.

For the NEO TIE, we implemented a fairly simple agent service message protocol for MPS using KQML, that allowed the scheduler to be called by another agent. Parameters of the scheduler could be set, and queries and updates to the internal data store of MPS (including scheduler results) could be responded

to. This simple API was subsequently extended greatly to support the TRIPS-CAMPS TIE, described below.

4. MIXED-INITIATIVE AGENT TEAM ADMINISTRATION (MIATA) WORKING GROUP

In January of 1999, Prof. Hendler, the incoming CoABS Program Manager asked Mark Burstein at BBN to lead the group of researchers interested in mixed-initiative agent systems within the CoABS program in a two year effort to demonstrate mixed-initiative agent systems dynamically interoperating with human users. This was entirely consistent with our initial vision for research on mixed-initiative agent teams. The working group adopted the name MIATA, for Mixed-Initiative Agent Team Administration. We began developing a plan for a simulation-based demonstration of large-scale mixed-initiative agent coordination that spring. As a first step in that direction, BBN began a six-month effort collaborating with James Allen and George Ferguson at the University of Rochester, that combined their interactive, multi-modal multi-agent TRIPS planning system with the agentized version of MPS. Thus, the CAMPS-MP agent continued to be an integral element of our experimentation and demonstrations of agent functionality and utility in the next series of TIEs, the MIATA TIEs.

4.1 THE ROCHESTER INTERACTIVE PLANNING SYSTEM - CAMPS

We developed a prototype mixed-initiative planning tool for airlift scheduling by integrating elements of TRIPS, The Rochester Interactive Planning System, an agent-based, interactive, mixed-initiative planning system using spoken natural language dialogue (Ferguson and Allen, 1998), with the CAMPS Mission Planner, an interactive airlift scheduling tool developed for the Air Force (Emerson and Burstein, 1999), together with some related resource management agents representing other parts of the airlift planning organization. See Burstein, Ferguson, and Allen (2000) for a detailed discussion of this work.

In developing this demonstration system, our approach was to extend our inital KQML agent service model for MP agents to reflect the complexity required to manage such a complex service through an intelligent interface agent, rather than directly through the systems native GUI. The GUI remained accessible to the user, but events happening throught the GUI now had to be monitored by the multi-model agent, and that agent (TRIPS) could invoke large portions of the functionality provided by the interface. As we did this, we collected and cataloged the reasoning and information interchange issues of the various agents forming the complete combined system. This included problems arising in interpretation and reformulation of user intent, and the planning of requests to the various back-end, airlift-domain-specific agents.

Overall, this TIE effort had a number of goals. First, we wished to demonstrate the relative ease with which the TRIPS agent architecture could be adapted to a new planning domain, and to interact with a new back-end planner. Second, we sought to understand what would be required for an effective agent ACL (Agent Communication Language) interface to a scheduling or planning tool that had its own GUI and was not developed for multi-modal mixed-initiative interaction. Third, we wished to develop a model for the problem-solving agent that could mediate between the user and a set of “back room” planning agents.

Our work involved the integration of three existing systems, TRIPS, MPS and an aircraft resource allocation system called the Barrel Allocator. TRIPS, The Rochester Interactive Planner System (Ferguson and Allen, 1998) is a multi-agent system that includes agents for speech recognition, natural language understanding, reference resolution, discourse management agents, speech generation, planning and plan recognition, among others. The TRIPS agents collectively provide a multi-modal interface by which users can discuss and develop plans through mixed-initiative interactions with what appears to

them to be a single 'intelligent' agent. The agents interact to produce this behavior by exchanging messages using the KQML (Finin, Labrou & Mayfield, 1993) agent communication language, operating in a hub-based architecture provided by the TRIPS Input Manager (TIM). Figure 1 shows the logical connections between the agents involved in the integrated TRIPS/CAMPS system.

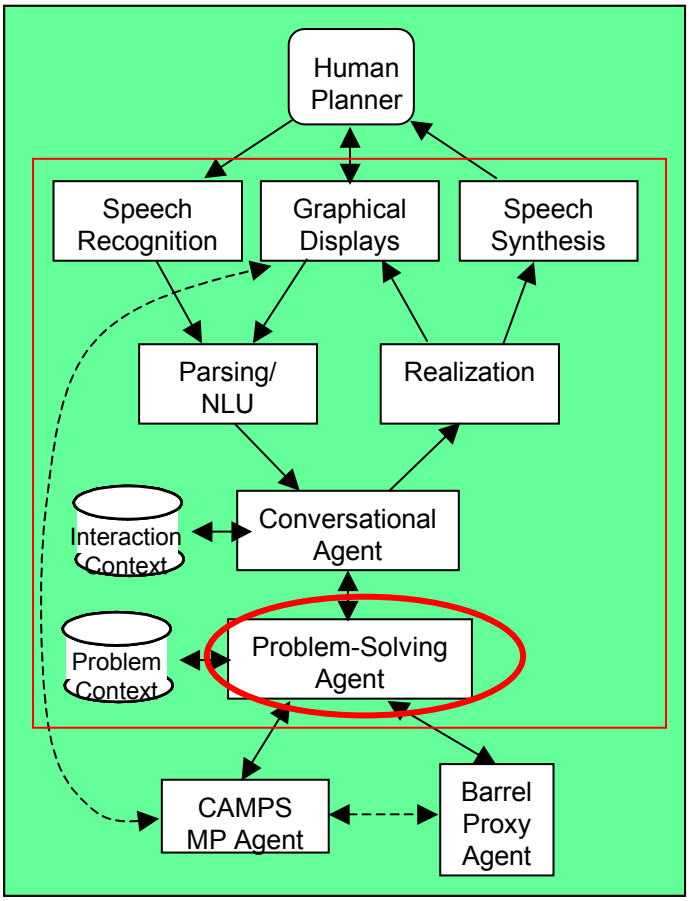


Figure 1: TRIPS/CAMPS agents

TRIPS-CAMPS supports a wide range of speech acts, ranging from direct requests (e.g., "show me the map"), questions ("Where are the transports?"), suggestions ("Let's use a helicopter instead."), acceptances, rejections, and a range of social acts.

The specialized back-end agents in TRIPS and earlier incarnations included an incremental (repair-oriented) temporal planner, a route finder, a scheduler, a simulator that represented the changing world state, and a temporal knowledge base agent. These various agents were invoked by the problem solving manager in response to user inputs. A typical interaction with TRIPS when working with the CAMPS-MPS system went as shown in Table 1. System responses are in bold face. All of the dialogue was verbal, and the TRIPS agents managed the conversation.

U: Show me the active plans.
T: Ok. *[Displays CAMPS list of plans]*
U: Let's work on the Georges plan.
T: Ok. *[Displays Georges plan on MPS GUI]*
U: Estimate resource needs.
T: 4 C141s and 1 C5 will be needed.
U: Where can we get them?
T: There are 1 C5 from Travis, 2 C141's from Charleston and 3 C141's from Mcguire.
U: Let's use two 141s from McGuire.
T: Ok. You still need 1 C5 and 2 C141's before we can schedule.
U: Use the C5 from Travis also.
T: Ok. You still need two C141's before we can schedule.
U: And use the 141s from Charleston.
T: 12 requirements could not be scheduled.
U: Show them to me.
T: Ok. *[Puts up list of failed requirements]*
U: Explain PWNC 1.
T: C5's can't land at Washington.
U: Lets use a C17 instead.
T: One requirement is still unscheduled.
U: Explain it.
T: C17's, C5's, and C141's cannot fly non-stop from Elmendorf to Roosevelt Roads.
U: Show me on the map.
T: Ok. *[Shows map]*
U: Use Charleston as an enroute stop.
T: All requirements are scheduled.

Table 1: Trip-Camps Dialog (U is User)

Figure 2 shows some of the graphical views as seen from a user's perspective. The MP agent, using pre-existing views, with one exception, generated the graphic displays. While a user could have accomplished much the same result by interaction with the CAMPS-MP GUIs, many steps would have required tedious entry of information through dialog windows. Furthermore, we were forced to extend the MPS' ability to explain scheduling failures in order to make the dialog more effective. These capabilities (without the speech) are part of MPS today, and are a key capability not found with many schedulers. That this kind of explanation was seen as critical first in conjunction with the TRIPS dialog system highlights how much the interaction modality effects system design.

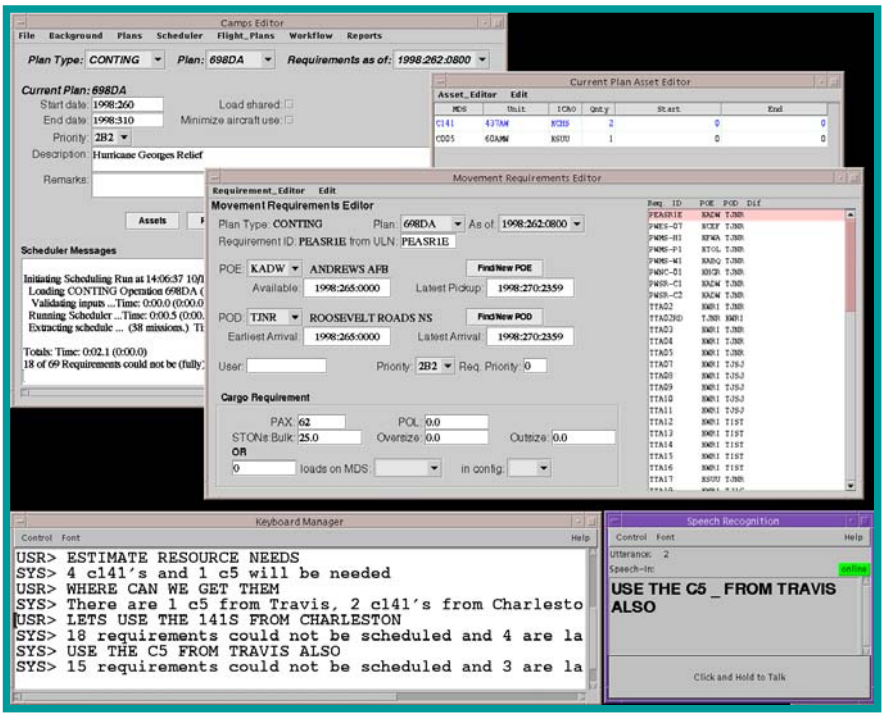


Figure 2: User views for TRIPS/CAMPS

This first-cut integration of TRIPS and CAMPS agents was developed in a relatively short period of time (approximately three months) in part because of the modular, agent-oriented construction of TRIPS itself. Aside from small efforts for extending the vocabulary and language understanding modules, the main effort was in 'agent-ifying' the airlift domain agents, CAMPS-MP and BARREL, and in developing a specialized problem-solver manager agent that could do the required interpretation and translation of user actions and airlift agent responses, in the context of the airlift mission planner's task.

We developed a KQML API that provided essentially a programmatic means of invoking the behaviors available through the scheduler's graphical user interface. This was done without explicit reference to the internal representations manipulated by and shared among the TRIPS agents. Table 2 shows a summary of the message signatures handled by the CAMPS-MP agent. It includes a set of commands for manipulating the various views provided by the CAMPS-MP interface, a set of commands for describing new problems and revising old problems to be posed to the scheduler, a request for invoking the scheduler, and queries for analyses of scheduler results and explanations of problems found in those results.

The main information passing messages that would be used also when another planning agent invoked MPS, instead of the user (NEW-PLANSET, REVISE-PLANSET, and AUGMENT-PLANSET) all take as keyword arguments the different kinds of data elements that define problems for the Mission Planner to solve. NEW-PLANSET is used to create a new problem from scratch, while AUGMENT-PLANSET is used to add ports, resources or requirements to an existing problem. REVISE-PLANSET is used to replace one of the previously specified sets of values.

We believe that this interface is representative of the kind of interface that one will get in practice when 'wrapping' a legacy system. As such we saw it as a reasonable target for the problem solver manager that it was to interact with. More interesting here was the need to not just wrap MPS but extend

its capability to produce explanations of why things went wrong, something that users of scheduling systems also require, but usually have to work hard to discover on their own.

KQML Performative	**CONTENT FORM**
REQUEST	(OPEN-VIEWER :VIEWER <VIEWER>)
REQUEST	(FOCUS-VIEWER :VIEWER <VIEWER> ...)
REQUEST	(CLOSE-VIEWER :VIEWER <VIEWER>)
REQUEST	(DISPLAY-TABLE :TITLE <STRING> :QUERY <SQL>)
REQUEST	(MAP-FOCUS :LATITUDE <LAT> :LONGITUDE <LON> :ZOOM <FLOAT> :PLAN-ID <ID>)
INFORM	(NEW-PLANSET :PLAN-ID <ID> :START-DATE <DATE> :END-DATE <DATE> :PRIORITY <PRI> :REQUIREMENTS (<REQUIREMENT>*) :AVAILABLE-ASSETS ((<ACTYPE><QTY><UNITID><START><END>)*) :AVAILABLE-PORTS ((<LOCID><NAME><ENROUTE?>...)*) ...)
INFORM	(REVISE-PLANSET ….*keywords same as NEW-PLANSET*….)
INFORM	(AUGMENT-PLANSET ….*keywords same as NEW-PLANSET*….)
ASK-ONE	(FLIGHT-PLAN :PLAN-ID <ID>)
ASK-ALL	(LIST-PLANSETS :FIELDS…)
REQUEST	(SHOW-PLANSETS :PLAN-TYPE <str> :NEW-SINCE <time> :CHANGED-SINCE <time>...)
REQUEST	(ANALYZE-PLAN :PLAN-ID <ID>)
REQUEST	(SHOW-UNSATISFIED-REQUIREMENTS :PLAN-ID <ID>)
REQUEST	(SCHEDULE-AIRLIFT :PLAN-ID <ID>)
REQUEST	(ESTIMATE-RESOURCES :PLAN-ID <ID>)
REQUEST	(EXPLAIN :OBJECT (UNSATISFIED :REQ-ID <ID>))

Table 2: Summary of KQML interface to CAMPS-MP

In integrating MPS with a framework for multi-modal mixed-initiative interaction, we have uncovered, or, in some cases, rediscovered, requirements for both the problem solver mediating between the user and these domain agents, and for the capabilities of the domain agents that will be suitable for participating in such systems.

In particular, we discussed in Burstein, Ferguson, and Allen (2000) the importance of information sharing between agents that reason and need to present information to the user, and the agents responsible for dialogue. We also provide an extended discussion of the important role that a capability to generate explanations can play in furthering the cooperative problem solving of the user+agents team. Section, 7: Cooperative Information Sharing details our subsequent research on ways to improve the information sharing amongst agents and their users and solutions for cooperative information sharing within organizations.

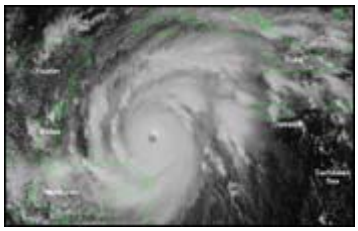
5. MIATA DEMONSTRATIONS

For two years (January, 1999 – January, 2001), BBN led the MIATA working group of researchers[1]

[1] The MIATA working group consisted of Mark Burstein, David Diller, Alice Mulvehill, Brett Benyo and Ed Pattison-Gordon (BBN Technologies), Karen Myers and David Moreley (SRI), George Ferguson and James Allen (University of Rochester), Sebastian Thrun and Jaime Schulte (CMU), Brian Drabble and Najam-ul Haq (CIRL, University of Oregon), Drew McDermott

within the CoABS Program. The main focus of the group was on the development of a series of simulation-based demonstrations of mixed-initiative human-agent computing in the context of a large organization. The goal was to explore and validate the potential for heterogeneous agent systems to support and function effectively within dynamic, distributed human organizations. We are convinced that it is only because we developed a large organizational simulation model operating in a realistic scenario that we were able to understand and address the issues discussed here and in Burstein and Diller (2003, in-press) and Burstein, McDermott, Smith and Westfold (2000a, 2000b).

We were driven by a scenario where users and software agents were teamed in a number of different offices representing the Joint Task Force (JTF) that coordinated the US Military's response to the Hurricane Mitch disaster. We built this scenario based directly on the historical record of the events from a number of sources, and information we were able to obtain from the government about the plans that were made by the US to airlift supplies to the region. Hurricane Mitch struck Central America from October 26 to November 4 1998, the second most devastating hurricane ever recorded in the Western Hemisphere. 200 mph winds and 75 inches of rain left a death toll of over 11,000, with thousands more missing, and more than three million people were left homeless or otherwise seriously affected. Mitch caused over $5 Billion in damages. Honduras was affected most severely, with widespread flooding and devastation of its transportation infrastructure and villages. At least 70% of Honduras' crops were destroyed.



The MIATA demonstration had six human users directing and interacting with over one hundred software agents to:

- Form teams and assign tasks,
- Gather intelligence about damage on the ground by interacting with a simulation, (MapleSim) of the region during and after the hurricane passed through,
- Plan for the deployment of relief supplies and the repair of roads and bridges,
- Manage logistical resources and the distribution of supplies,
- Report and respond to problems 'in the field'.

The software agents used were a mix of 'agent-wrapped' versions of pre-existing software tools and agents developed specifically for the demonstration. The 'wrapped' agents were logistics planning and scheduling systems[2] of various kinds with graphical user interfaces developed to address the needs of users with military logistic planning tasks. The other agents in the model were developed in OMAR (Deutsch, 1998) and PRS (Myers, 1993), two very similar reactive procedural execution systems. They were mostly representing field agents (trucks, helicopters, aircraft) and their local commanders (truck, helicopter, and airlift wing company commanders, other staff agents on teams). Agents communicate with users and other agents within the military organization, as well as agents representing other non-governmental organizations like the Red Cross. We used several different agent communications

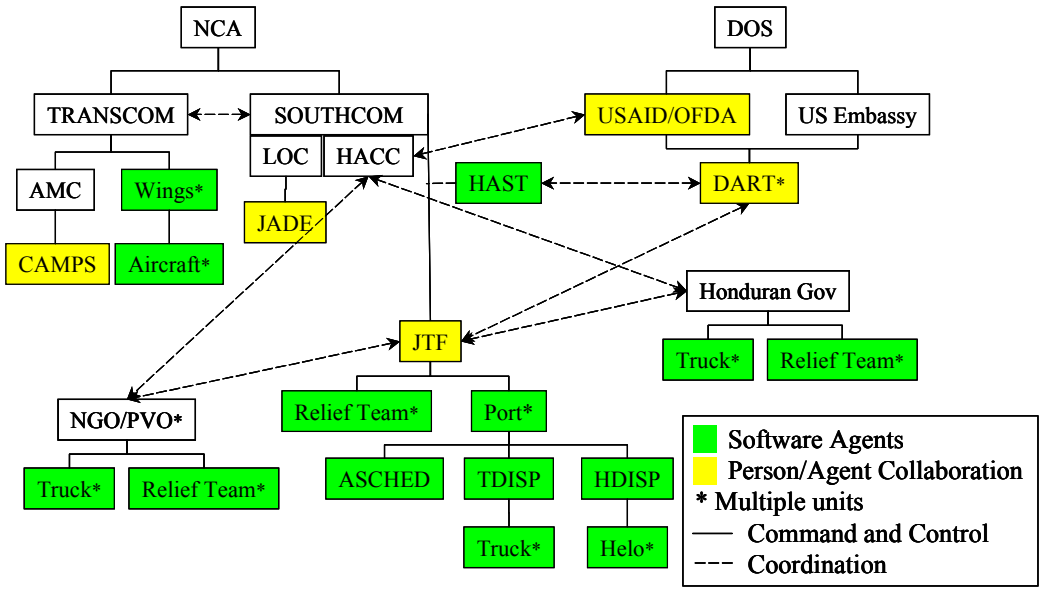
(Yale University), Doug Smith and Stephen Westfold (Kestrel Institute), and Steve Ford (OBJS, Inc.). We are indebted to everyone for their hard work on this project.

[2] The CAMPS Mission Planner (Burstein & Emerson, 1999) is a scheduling system for cargo aircraft being developed by BBN and Kestrel Institute for the Air Force. The JADE logistics planner (Mulvehill & Caroli, 1999), developed at MITRE and BBN, is a prototype interactive case-based reasoning tool for composing deployment plans consisting of lists of cargo to be picked up from depots and delivered to the theater. Both were initially stand-alone systems with graphical user interfaces that were adapted for this work. CIRL's DEO scheduler was wrapped as an agent without a user interface to provide the functionality required to schedule the deliveries and other activities of trucks and helicopters.

mechanisms for different clusters of agents, which were made to interoperate via the CoABS GRID[3], an agent interoperation framework defined on top of Jini (Arnold, et al., 1999).

5.1 MIXED-INITIATIVE MANAGEMENT OF A JOINT TASK FORCE

Our simulation of the US Military's disaster relief effort in response to Hurricane Mitch consisted of a hierarchical set of teams with humans (with agent assistants) leading the highest level teams. Figure 3 illustrates the organization structure of the various organizational entities that were involved in the real events. In addition to using agents to represent different groups within the military organization, we also modeled interactions with other governmental organizations such as the Honduran government, as well as non-governmental organizations like the Red Cross, which was also providing medical relief. At the base of the simulation, we have the 'mobile' field agents that act through messages sent to the MapleSim simulation of the ground state in Honduras. MapleSim, developed at CMU modeled the hurricane passing through the country, and probabilistically created the damage to be repaired. It also supports messages sent to towns to collect state information, limited by communications damage, or visibility distance (for trucks and helicopters).



AMC Air Mobility Command	HACC humanitarian assistance coordination center	LOC logistics operation center	SOUTHCOM Southern Command
ASCHED asset scheduler	HAST humanitarian assistance survey team	NCA National Command Authorities	TDISP truck dispatcher
CAMPS combined air mobility planning system	HDISP helicopter dispatcher	NGO nongovernment organization	TRANSCOM Transportation Command
DART disaster assistance response team	JADE joint assistant for development and execution	OFDA Office of Foreign Disaster Assistance	USAID U.S Agency for International Development
DOS Department of State	JTF joint task force	PVO private voluntary organization	

Figure 3: MIATA Organizational Model

Figures 4 and 5 show a pictorial and timeline representation, respectively, of the users, agents and the communication flow between them in our demonstration framework. The main body of the team is a Joint Task Force (JTF), a dynamically formed organization structured by doctrine with a set of roles for different sub organizations. The main teams reporting to the commander are an intelligence collection group (JTF-J2), an operations group (JTF-J3) and a logistics group (JTF-J4). The task force communicates with other elements of the larger US Military organization. The JTF Commander reports to a US Joint Commander (not shown), whose role in the simulation is to give the initial guidance to form

[3] GITI/ISX Grid Vision Team, "The CoABS Grid: Technical Vision," http://coabs.gobalinfotek.com, 2000.

the team, and to receive progress reports. The JTF logistics team works with the logistics arm of the Joint command and, indirectly, with the US Transportation Command, which creates schedules for the airlift of supplies that are executed by the wing teams.

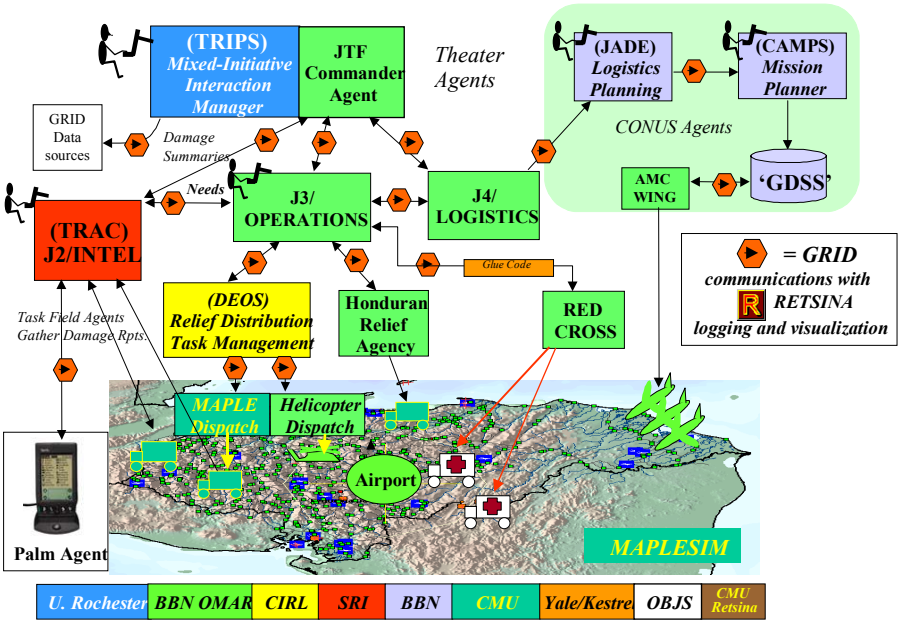


Figure 4: Users and Agents in MIATA Simulation

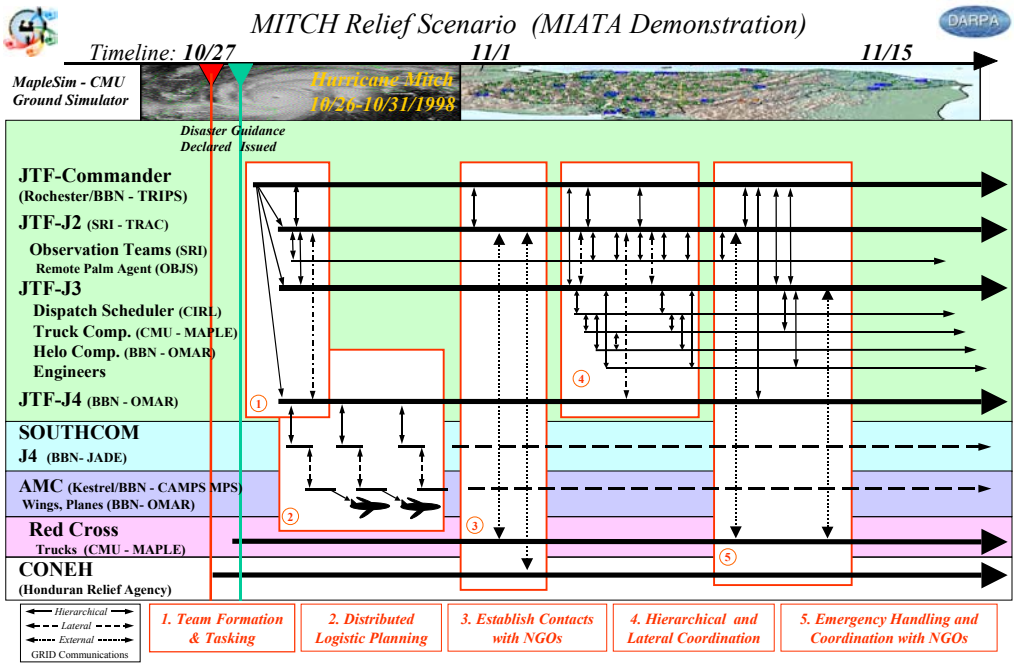


Figure 5: MIATA Relief Scenario Timeline

Collectively, the Joint Task Force team of humans and agents formed a hierarchical organization that interacts with the simulation to perform a variety of basic relief tasks such as the distribution of food,

water and shelter, providing medical relief, and repairing infrastructure such as roads and bridges. As part of the demonstration, we developed a number of classes of agents, each defined generically by its functional role in the organization. At the lowest level, the simulation includes a large number of field agents, such as trucks and helicopters, engineers, and medical relief teams. These agents act like 'drivers' of vehicles in that they interact with the MapleSim simulator (See Figure 6) to move simulated vehicles about, gather information (by messages received back from MapleSim), and can then generate field reports to their team leaders, and change the state of the simulation world.

A second class of agents, Task Management Agents (TMA), schedule, task, and monitor other agents. They are the leaders of the field agent teams. Often these agents are also used as 'information collectors'; agents that collect and disseminate (on request or periodically) information about field conditions for other agents on their teams. Typically, TMAs manage field agents, such as the helicopter and truck company commander agents that act as dispatchers for their vehicle agents, the wing commander agents that manage groups of cargo aircraft.

Finally, a third class of agents, Personal Assistant Agents (PAA), supports mixed-initiative interactions between agents and humans. Work to date has focused primarily on three Personal Assistant Agent clusters: A cluster supporting the Joint Task Force (JTF) Commander, one supporting the JTF Intelligence (J-2) staff and one supporting the Operations (J-3) staff.

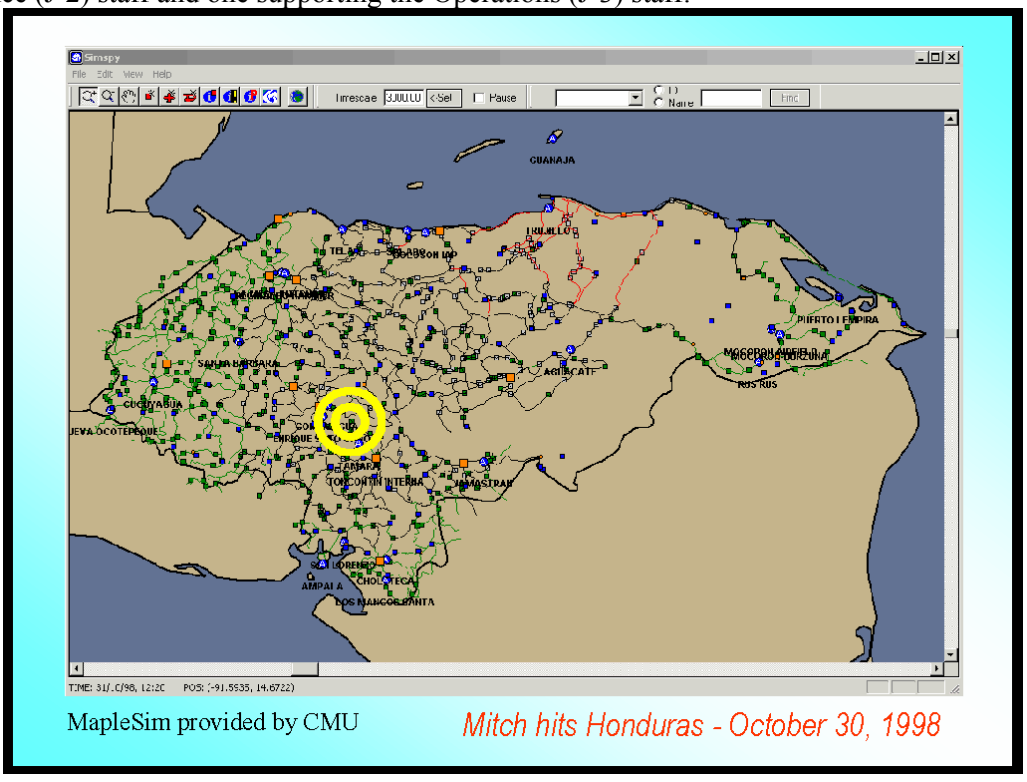


Figure 6: MapleSim with hurricane crossing

5.1.1 TASKABLE AGENT INTERFACE FOR COMPUTER HUMAN INTERACTION

In a complex system involving multiple semi-autonomous agents, it is critical for human commanders to be able to easily monitor and control the agents under his or her command. A commander must be able to respond to unexpected events that the agents are unable to handle, or to retask the agents due to a change in mission goals or newly received information. In order to control a semi-autonomous agent, a

commander needs to be able to monitor the agent's progress and observations, so that the agent's current operating environment is understood. In addition the commander needs to be able to redirect the agent by modifying its goals.

We created a multi-modal commander's agent interface called TAICHI (Taskable Agent Interface for Computer Human Interaction) that gives a commander this sort of control over a multi-agent system.

5.1.2 AGENT MONITORING

In order for TAICHI to provide the commander with a detailed picture of what an agent is doing, it needs access to status information from that agent. This information is provided through status event generation goals incorporated into our agent framework. Any time an agent starts, succeeds, fails, suspends, or resumes a task, a status event is generated and sent as an agent message to TAICHI. The event generation code built into our agent model by use of aspect-oriented programming (Kiczales, 1996) via AspectJ[4] for Java agents or around methods for Lisp agents. TAICHI, implemented as an agent itself, receives these status events through the agent communication mechanism in use, such as the CoABS Grid, and displays the events graphically in its task monitoring display. If an anomalous event occurs, the system may alert the user verbally to the issue indicated on the display. In the top left panel of Figure 7, for example, TAICHI is displaying the status of DOD-COMMANDER-1, an agent that has five assigned tasks. Four tasks, the three establish-team-members, and the launch-team, have been marked with a black dot to show that they have completed successfully. The fifth task, perform-assessment, is ongoing, and has been delegated to the agent, INTEL-AGENT-1.

In addition to task status information, TAICHI can display location information on an OpenMap™[5] map display. This can give the commander a visual representation of the location of an agent with a physical manifestation, and can be used to display domain specific data. The map display in Figure 7 shows cities, bridges, and the road network essential for relief operations in Honduras in the MIATA scenario.

[4] http://aspectj.org

[5] http://openmap.bbn.com/

5.1.3 AGENT TASKING

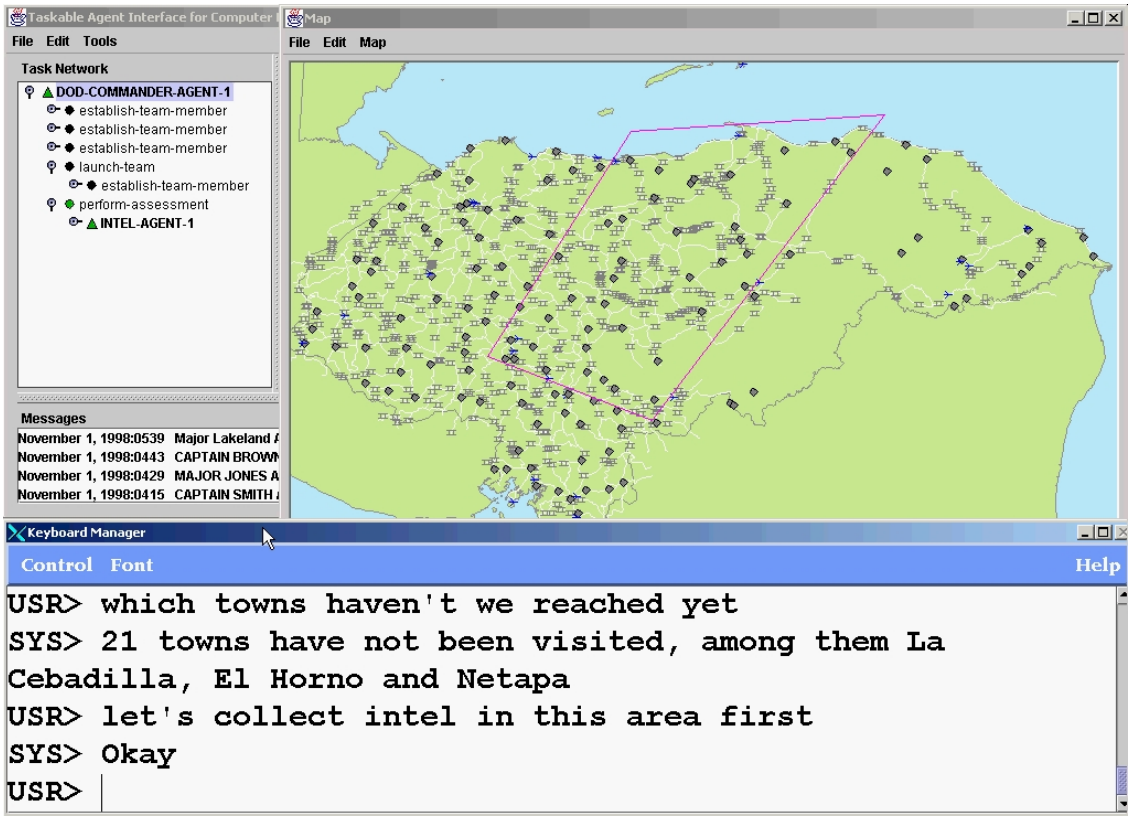


Figure 7: TAICHI display for the MIATA scenario

Once the commander has a full picture of the tasks the agents are executing and their status, the commander may wish to assign new tasks, or to stop execution of existing ones. TAICHI is a multi-modal interface, giving the commander a number of different ways to provide input. The bottom panel of Figure 7 shows text and speech input. TAICHI uses the TRIPS or the BBN HARK Recognizer™[6] to understand a set of domain specific sentences. The commander can type the sentences in as text, or speak them into a microphone. In Figure 7, the commander first asked what towns haven't been visited yet. TAICHI parsed this sentence, recognized it as an information query, and sent the query to the intelligence-gathering agent, INTEL-AGENT-1, that was assigned to the commander's agent team. In this case, the intelligence agent responded with a list of 21 towns, which were highlighted on the map. The query result was also printed out to the dialog window(bottom), and spoken out loud.

Another method for the commander to provide input is to use the map drawing tools to select map objects or draw regions. In figure 7, the commander has drawn a polygon on the map, encompassing the 21 towns, returned in the previous query, that have not been visited yet. The commander then tasked the intelligence-gathering agent to gather data about the status of the towns inside the drawn region by saying "lets collect intel in this area first" into the speech interface. This illustrates our simultaneous use of speech and gesture to convey information to the system.

[6] http://www.bbn.com/speech/bbnhark.html

5.1.4 TEAM TASKING AND FORMATION

In order to develop mixed human and agent organizations that are both flexible and robust, we have developed agents with explicit models of the team organization, including agent capabilities, roles, and lines of authority between agents based on their agreed-upon roles (Burstein and Diller, 2001, Forthcoming; Burstein, Mulvehill, and Deutsch, 1999a, 1999b). Similar model-based approaches have been explored by a number of other researchers (e.g., Cohen, Levesque, and Smith 1997; Rich and Sidner 1997; Tambe and Zhang 2000). In order to enable mixed-initiative team formation, this information must be useful to agents interacting with humans. To aid in this, we have developed, in conjunction with the University of Rochester (Ferguson and Allen 1998), a multi-model interface designed to facilitate human interaction with Personal Assistant Agents (PAAs).

As part of our disaster relief scenario, the JTF Commander forms the supporting JTF staff of humans and agents through a spoken dialog with his PAA in order to determine what other humans and their supporting PAAs are available to perform a number of critical roles such as Intelligence (J-2), Operations (J-3), and Logistics (J-4) as part the JTF team. A simple example of the interaction between the JTF Commander and his supporting PAA is as follows:

Human: Establish a Joint Task Force at Soto Cano.
Agent: Alright
Human: Show me the officers there.
Agent: <Displays a table of officers and their areas of expertise. >
Human: Assign Captain Smith to be the J2.
Agent: Ok.
…
Human: Show me the team objectives.
Agent: <Shows objectives relevant to this task force.>
Human: <Selects objectives> Inform the team.
…

Agents asked to perform a particular role within a team have the opportunity to either accept or reject the role, dependent upon the agent's ability to perform the role, as a function of their existing capabilities, any pre-existing roles that may conflict with proposed role or team, or the overriding authority of a human controller. Furthermore, agents should be able to accept a role 'with qualifications'; choosing only to perform a subset of its capabilities as a member of that team. Upon formation of the team, all team members are informed of all other team members, their roles, and any capabilities or command and control deviations from the default model.

Teams formation can occur either through a mixed-initiative operation between human and PAA or an autonomous operation by a PAA or Task Management Agent. In our disaster relief scenario, while the upper echelon of the JTF was formed with interaction from the human JTF commander, supporting subteams, including intelligence and operations teams, were formed autonomously by each team's lead agent. These supporting subteams were formed by agents as required to support the tasks accepted by the JTF team leader.

The interaction between agents or humans and agents regarding the tasking of subordinate agents requires a sequence of messages or communications policy (e.g., Bradshaw, et al. 1997), starting first with a request to perform a task. The agent's decision to accept or reject a task is a function of a number of factors, including the capabilities of the agent, the tasking agent's authority over the tasked agent, the existing goals of the agent, and other preexisting tasks. Acceptance or rejection of the task is reported, and the task is queued for execution, upon receiving an execution message from the tasking agent (See Figure 8). Upon confirmation of task execution, the agent determines any necessary conditions for completing the tasking, including the formation of subteams and delegation of subtasks. By default, task

failures or completions are reported to the tasking agent. However, the tasking agent may override the default reporting actions, specifying the results be reported to another agent. For example, the JTF Commander may request the J-2 perform a survey of the central region of Honduras and to report the results to the J-3. Advice (Myers 1996) can be used to say when a situation should be reported immediately versus periodically, and in a detailed or summarized form.

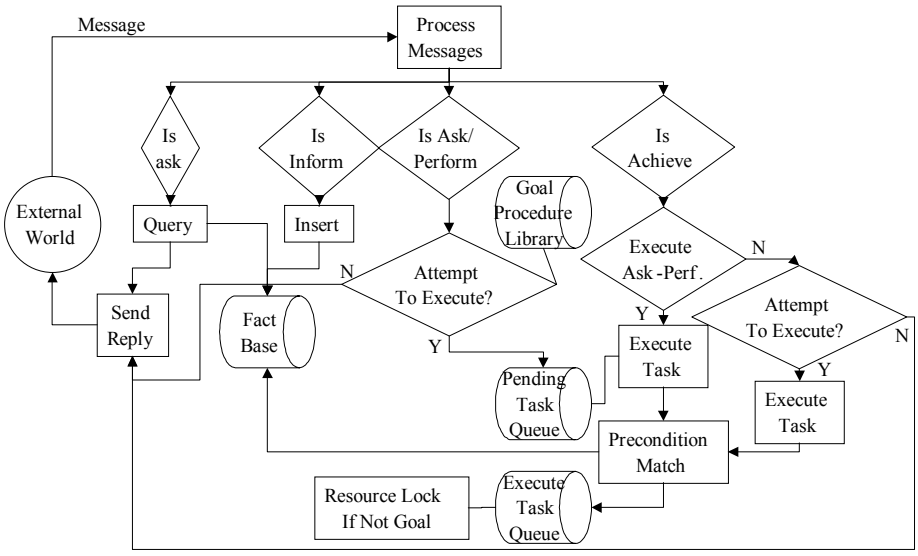


Figure 8: Message Processing

6. TRANSLATION

Agents that were not designed to work together may not use the same representations. Thus, getting agents to communicate often requires translating the data structures of the sender (the source representation) to the format required by the receiver (the target representation). Assuming that there is a formal theory of the semantics of the two formats, which explains both their meanings in terms of a neutral topic domain, we can cast the translation problem as solving higher-order functional equations. Some simple rules and strategies apparently suffice to solve these equations automatically. The strategies may be summarized as: decompose complex expressions, replacing topic-domain expressions with source-domain expressions when necessary. A crucial issue is getting the required formal theories of the source and target domains. We believe it is sufficient to find partial formalizations that grow as necessary.

BBN collaborated with their subcontractor Yale University and Kestrel Institute on research aimed at getting agents to communicate with each other when their internal ontologies were different. By 'agent' we mean programs that operate at a high enough semantic level that they can form new connections to other programs in order to get a job done. To make such a connection, an agent must find other agents that might carry out a task on its behalf, and then establish a dialogue with them. Several researchers have examined facets of this interchange, including how agents might search for each other (Sycara et al., 1999), how they might communicate once they have linked (Martin, Cheyer, & Moran, 1999), and what 'speech acts' they might employ (Finin, Labrou, & Mayfield, 1997). However, the most pressing problem is getting them to speak the same language. This is our focus here.

Suppose one agent, *A*, needs a certain fact, and agent *B* can supply it. Assuming that some previous 'brokering' or 'advertising' phase has brought the two agents together, there remains the problem that the way *A* represents facts and the way *B* represents them are probably not compatible. It is necessary to

interpose a translation program between the two. We call this glue code. The problem is to generate glue code automatically.

For example, suppose we are given a scheduling agent *MP* that produces an airlift schedule. For each aircraft, the schedule gives the sequence of flights that it is scheduled to make. In another agent (the Barrel Allocator), we also have an alternative representation of aircraft schedules that show when they are "committed" to a mission. This model requires a table that specifies, for each time slot, the number of each type of aircraft that are committed (i.e. not free for allocation) in that time slot. We were able to use models of the data or knowledge used by these two agents to automatically derive a translator function *f* from schedules to numbers of committed aircraft. The approach is quite general to structured knowledge representation formalism. It is based on combining some very general purpose rules of translation expressed in a higher order logic with ontologies for the source and target domains and knowledge of shared abstractions of the concepts represented in those domains.

Details of our approach for automatically deriving the "glue code" programs for translating the output of a source agent to the input representation of a target agent can be found in Burstein and McDermott, 1996; Burstein, et al, 2000a, 2000b; and McDermott, Burstein and Smith, 2001. This work has continued under the DARPA DAML Program.

7. COOPERATIVE INFORMATION SHARING

As part of our interest in models of and support for mixed-initiative human control of software agent teams, especially in the larger context of dynamic, real world organizations, we have developed a model for the establishment of cooperative information sharing among agents on teams formed dynamically for particular purposes within such organizations (Burstein & Diller, 2003, Forthcoming). We argue that effective information sharing in the presence of such teams requires the active dissemination of descriptions of current and future information needs to both local teammates and to the larger organization. Only by this mechanism can one avoid having to make explicit at design time who will provide each bit of the information. We consider how information sharing within the organization can be promoted not only for the immediate goals shared by a tightly coordinated team, but some of the likely information needs of the larger organization going forward. We illustrated the model in the context of the MIATA TIE.

The model proposed is based on representing and disseminating knowledge about information needs and information provision capabilities of agents. These agents are assumed to exist as part of a dynamic organization composed of interrelated heterogeneous agents teams and humans supported by agents. Using this model, agents, when given specific tasks, can determine whom to tell about their status, failures, and information that they discover that is potentially needed by others on their team or in the larger organization. An important goal of the model is that it be flexible enough so that users directing teams can initially make assumptions about agents' policies for information sharing, but change the information flow behavior dynamically.

A key element of the approach is the announcement by each agent of information requirements and anticipated future capabilities to do information provision, that is, the kinds of information the agent may come to have as a result of its intentions. These announcements to teammates establish the conditions for cooperative information sharing. Furthermore, by associating information policies with default intentions of team roles and also the capabilities associated with roles that may be activated by team plans, agents can quite easily initiate dialogs to coordinate information sharing when accepting roles or initiating new tasks.

A key issue for human and mixed human/agent organizations are the policies for information flow to support the varied tasks of individual agents and groups within those organizations. When considering how agent teams might support such policies from the perspective of shared intentions or shared plan theories, there are several difficulties. First, while these models provide some of the theoretical basis for

motivating information flow, in terms of cooperation rules or policies to support the objectives of teammates, it may be difficult or impossible to implement these abstract policies directly in software agents with limited inferential capabilities, as is frequently the case in heterogeneous agent systems. Second, the informational needs of agents can be arbitrarily context specific, which means that all team agents must know more about each other's detailed plans *and their knowledge state* than even a Full Shared Plans model (Grosz & Kraus, 1996, 1999) would require.

We argue that the determination of what information a teammate agent will need to achieve its announced intentions should, wherever possible, be done by either or a combination of the following:

- making a characterization of that information need explicit in the shared model of that agent's capability (to intend that result), when it is announced at team formation,
- having the agent announce a specific information need when it commits to achieve that objective.

The key observation here is that information needs should be made explicit, rather than left to shared knowledge of planning operators. It is frequently assumed that a capabilities description for an agent, much like APIs in more traditional distributed object systems, are composed primarily of the set of message patterns that an agent will respond to, and perhaps a characterization of the response. Yet stating that an agent can respond to a query containing a variablized well formed formula provides almost no information about what tasks or queries it can respond to.

To address this problem, we identify two kinds of communication about information to be advertised by all agents in an organization. These advertisements are associated with either the basic intentions associated with acceptance of a role or with specific intentions adopted by an agent in support of individual or teams goals:

Information Provision (IP) advertisements declare that the agent sending the message has an intention to achieve some purpose or execute some plan that results in it having information of the specified type. This signals an intention that it will answer queries with the identified classes of content, or provide such information on receipt of a subscription request. Information may be requested either on a periodic or 'as learned' basis. Information Provision advertisements are not retracted unless the original intention is aborted. IP advertisements are denoted as either Active (IPA) or Passive (IPP), depending upon whether the agent is actively pursuing the acquisition the information, or merely serving as a passive, but opportunistic gatherer and provider of that information.

Information Requirements (IR) advertisements declare that the agent sending the message has an intention to achieve some objective requiring the information. The requirement may be either for the purpose of planning how to achieve the intent, execute a conditional plan, or for use during execution (i.e., processing the information).

Simultaneously advertising an IR and IP over the same class of content suggests the agent's role as a 'knowledge source' for that information in the future. The agent will both collect the information from anyone who provides it and provide the information when needed to others. Furthermore, some 'knowledge sources', like the J2 in MIATA, are active sources, in the sense that they will perform actions to acquire the information (or direct teams that do so). Such agents issue IPA advertisements, which further indicate that they may adapt existing plans acquire content they do not have at the time queried in order to acquire the information sooner.

IP and IR advertisements associated with an operation are immediately announced to team members upon the agent's intention to perform that operation. An agent, upon learning of an IP advertisement that matches the class of content it requires, may initiate a subscription protocol with the agent advertising the IP, enabling it to automatically receive the desired information upon its availability, or with a specified delivery schedule. The subscription request details the specific information desired, the rate at which the information should be delivered (e.g., on-occurrence, on-completion, or periodic with a given period,

such as every once a day), and the level of detail to be provided (e.g., full vs. a specified summary level). Subscriptions may be removed after an information requirement ceases to exist.

In order to support information sharing through IP and IR advertisements, an information sharing protocol has been established to facilitate the dispersal of advertisements within and among teams. In MIATA, these teams are organized hierarchically, though that is not critical to our model. What is important is that agents are often members of more than one team. For example, the Operations (J3) agent is a member of the JTF staff, but also is leader of the subordinate Operations team. These multi-team agents enable the sharing of information across teams. A Liaison agent may be used, often in conjunction with a Translation agent, to facilitate communication and information sharing between organizations with no common members. In MIATA, the Red Cross and the JTF teams communicate via such an agent. We have developed a number of rules that govern dissemination of information advertisements between team members, and their forwarding to adjacent teams. A subset of these is shown in Figure 9.

Rule 1. An agent (x), after accepting a role (R) on a team, tells all teams, of which it is a member, the information it can provide (IPs) as a function of its default intensions (I).

$$\forall x\,[\forall k \ni x \in T_k\,[\forall y \neq x \in T_k \supset \text{tell}(x, y, IP_I)]]$$

Rule 2. An agent (x), after accepting a role (R) on a team, tells all teams, of which it is a member, of any information requirements (IR) associated with its default intentions (I).

$$\forall x\,[\forall k \ni x \in T_k\,[\forall y \neq x \in T_k \supset \text{tell}(x, y, IR_I)]]$$

Rule 3. After an agent (x) completes an intention (I), it retracts all IRs associated with that intention from all team members.

$$\forall x\,[\forall k \ni x \in T_k\,[\forall y \neq x \in T_k \supset \text{tell}(x, y, \neg IR_I)]]$$

Rule 4. Upon generating an intention for a capability (C) an agent (x) tells all team members of any IRs associated with that capability.

$$\forall x\,[\forall k \ni x \in T_k\,[\forall y \neq x \in T_k \supset \text{tell}(x, y, IR_C)]]$$

Rule 5. After an agent (x) completes an intention formed for a capability (C), it retracts all IRs associated with that capability from all team members.

$$\forall x\,[\forall k \ni x \in T_k\,[\forall y \neq x \in T_k \supset \text{tell}(x, y, \neg IR_C)]]$$

Rule 6. An agent (y) shares all IRs it is told about with all teams it is a member of, except the teams from which the originating agent (x) is a member.

$$\forall y[\text{tell}(x, y, IR) \supset \forall k \ni y \in T_k \wedge x \notin T_k\,[\forall z \neq y \in T_k\,[\text{tell}(y, z, IR(x,\tau))]$$

Rule 7. An agent (y) informs all team members of any IR retractions it is told about.

$$\forall y[\text{tell}(x, y, IR) \supset \forall k \ni y \in T_k \wedge x \notin T_k\,[\forall z \neq y \in T_k\,[\text{tell}(y, z, \neg IR(x,\tau))]$$

Rule 8. If an agent (x) has a belief ($\bar{\tau}$) that matches the information content of an IR from another agent (y), agent x tells agent y belief $\bar{\tau}$, provided agent x has a means for communicating with agent y, otherwise agent x tells the agent (z) which had communicated the IR.

$\forall x[\text{bel}(x, \overline{\tau}) \wedge \text{tell}(z,x,IR(y,\tau)) \wedge \text{match}(\overline{\tau}, IR(y,\tau)) \supset$
IF comm(x,y) THEN tell(x,y, $\overline{\tau}$) ELSE tell(x,z, $\overline{\tau}$)]

Figure 9: *IP* and *IR* dissemination rules.

Rules 1 and 2 support the sharing of *IP*s and *IR*s associated with basic intentions that are a part of the acceptance of a role. Upon acceptance of a role, an agent shares all *IP*s and *IR*s associated with that role with all team members. Rules 4 and 5 govern the sharing of *IP*s and *IR*s associated with an agent's capabilities. Upon acquiring an *intention to* described by an agent capability, the agent shares the associated *IP*s and *IR*s with its team members. Rule 6 specifies that *IP*s and *IR*s should be 'passed on' to other team agents even if the agent is not the originator of the advertisement. Rule 8 governs whom an agent should inform if and when it acquires information pertinent to an *IR*. Finally, rules 3 and 7 allow for the retraction of *IR*s upon completion of an intention. *IP*s are not normally retracted, as agent may continue to provide information. Additional rules (not shown) are required for determining when and how *IR*s should be combined. Ultimately, organizational workflow analyses may be needed to discover when it is advantageous for an agent to become an information collector/provider for a team of agents with intersecting information requirements.

We have presented a framework for dynamic information sharing among teams of humans and agents working as parts of larger, frequently hierarchical, organizations. The model was motivated by our work to develop approaches to mixed-initiative human/computer control of agent-based systems, and in particular from our role in the development, in coordination with a large team of other researchers, of the MIATA demonstration of mixed-initiative agent teams in a hurricane disaster relief scenario. The key observation is that information needs, as well as agent capabilities to achieve classes of intentions must be shared among teams and across organizations. In many cases, the information sharing needs and provision capabilities need wider dissemination than the sharing of team intentions. By using an approach where the announcement of intentions is accompanied by information about relevant information needs and expected information acquisition activities, we can create an environment where agents can develop their own information flow models dynamically. This is critical to mixed-initiative command and control of such systems as it removes the burden from users who might otherwise need to explicitly characterize which agents needed to communicate to achieve team objectives.

8. COALITION AGENTS EXPERIMENT

The Coalition Agents Experiment (CoAX) TIE relies upon an unclassified fictitious military scenario named Binni (Rathmell, 1999) that was created to experiment with coalition military operations. Binni is set in the year 2012 and involves three imaginary countries in Africa – Binni, Gao, and Agadez. Due to a conflict in the region between these three countries, a multinational UN peacekeeping force is brought in stop the conflict. The multinational force includes the United States, the United Kingdom, Canada, and Australia. During the course of the scenario, a fourth imaginary country – Arabello – is called upon to join the coalition. The Binni scenario provides a rich and militarily-relevant setting for experimenting with agent-based systems for coalition operations.

One of the critical concerns in any coalition operation (military or civilian) involves protection of sensitive or proprietary information. The Binni scenario models the complexities and nuances of the relationships between different countries that make up the coalition peacekeeping force. For example, the US, UK, and Australia have a high degree of mutual trust whereas Gao, which is also a member of the coalition, is trusted to a lesser extent. Therefore, from the perspective of one country (such as the US), there are three different scopes for information sharing: agents that are part of the US, agents that are part of the UK and Australia, and agents that are part of Gao.

Moreover, Gao starts out as a member of the coalition force but is then found to be providing misinformation to the rest of the coalition in order to advance its own private agenda. When this deception is discovered, the trust relationships are altered and the degree of information sharing between agents has to be adapted accordingly.

Finally, during the scenario Arabello joins the coalition and new trust relationships must be established between the existing coalition members and Arabello. The coalition needs to be able to effectively manage their concerns about information released by its agents to Arabello's agents and vice versa.

8.1 SIMWORLD

In order to more realistically model actors and actions occurring in the CoAX scenario, we used a simulator developed at BBN, SimWorld, for modeling the movement and interaction of agents and objects in the CoAX domain. SimWorld provides software agents with a simple world of objects, space, and events with which to interact. SimWorld provides an object hierarchy enabling the representation of passive objects, such as cities, roads, and bridges; active objects, such as sensors, transmitters, and vehicles with scripted motion; and agent avatars, that is, objects that are controlled by software agents from outside SimWorld.

SimWorld was used in the COABS/COAX demo (see Figure 10) to maintain ground truth; that is, where naval vessels of all sides were located at each moment of the simulation. SimWorld objects represented cities, ports, airbases, vehicles, sensors, and weapons systems. The Arabello underwater acoustic sensor network was implemented as a collection of SimWorld objects. A software agent monitored the reports the sensors sent of their detections. Figure 10 shows a passive acoustic sensor (small circle with a dot at the center) detecting the presence of two submarines (red). Three other passive acoustic sensors have detected a ship (green). The ship is carrying its own active sensor (large circle), but no vessels are currently within range.

Software agents interact with SimWorld using one of two APIs. Both APIs allow agents to add, remove, monitor, and control SimWorld objects and to discover other objects that are in a simulation. Since SimWorld is implemented in the Java programming language, the first API is the set of Java objects and their methods that are defined in SimWorld and can be called from a Java program. The second API is a set of messages that can be sent to SimWorld. SimWorld can send and receive messages using sockets, Java's Jini technology, or the COABS Grid (http://coabs.globalinfotek.com). Message content can be either in the FIPA ACL string format[7] or in a FIPA ACL inspired Java object array format.

[7] http://www.fipa.org/specs/fipa00070/

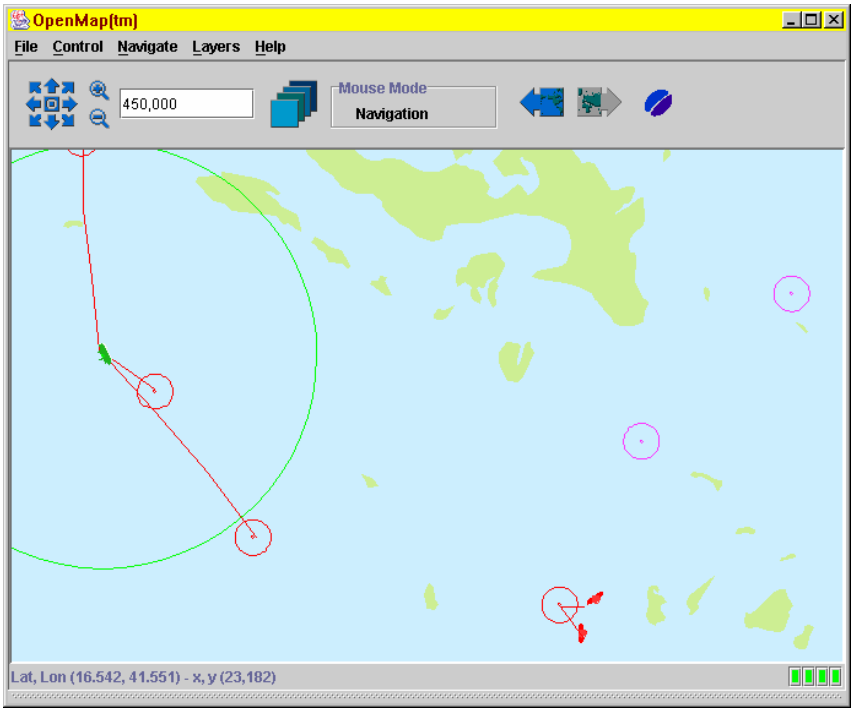


Figure 10: Sensor detection in the COAX simulation.

SimWorld is implemented using OmarJ[8], an open source simulator developed at BBN. OmarJ provides the simulation clock, messaging infrastructure, and scenario control: selecting which scenario to run, starting, pausing, and resuming the scenario, and setting the speed at which the scenario runs. SimWorld supports using OpenMap[9], an open source Java toolkit developed at BBN for displaying geospatial information, as a means for visualizing the execution of scenarios. In OpenMap, geospatial data from a given source is displayed using a "layer" and multiple layers can be shown at the same time to create a display that combines geospatial data from different sources. In addition to showing the location and movement of simulation objects, the SimWorld layer supports the editing of scenarios. Object properties can be viewed and changed and new objects can be added by dragging them from a palette and dropping them on the map. Scenarios can be saved using an XML file format.[10].

8.2 INFORMATION SHARING IN THE COAX TIE

8.2.1 DYNAMIC INTEGRATION OF AGENT SERVICES

One of the most difficult problems for an agent interacting with another dynamically discovered agent is translating the syntax and semantics of the agent communication languages. The traditional solution involves a human programmer coding a specific interface. The time and effort required by this approach can severely limit the usefulness of these dynamically discovered agent interactions. For example, in the CoAX scenario, an intelligence gathering agent, from the newly joined coalition member Arabello, registers its capability to provide contact reports from its underwater sensor grid. The coalition agents that

[8] http://omar.bbn.com/

[9] http://openmap.bbn.com/

[10] http://www.w3.org/TR/REC-xml

are searching for an enemy submarine need to be able to access, understand, and integrate the Arabello information feed as fast as possible in order to neutralize the enemy submarine before more damage can be done. Use of the DARPA Agent Markup Language (DAML) and DAML-Services (The DAML Services Coalition, 2002) allows the agents to perform many of the translation tasks, previously performed by a human programmer, themselves. Figure 11 illustrates the advertisement of Arabello capabilities and the discovery and usage of those capabilities by US agents.

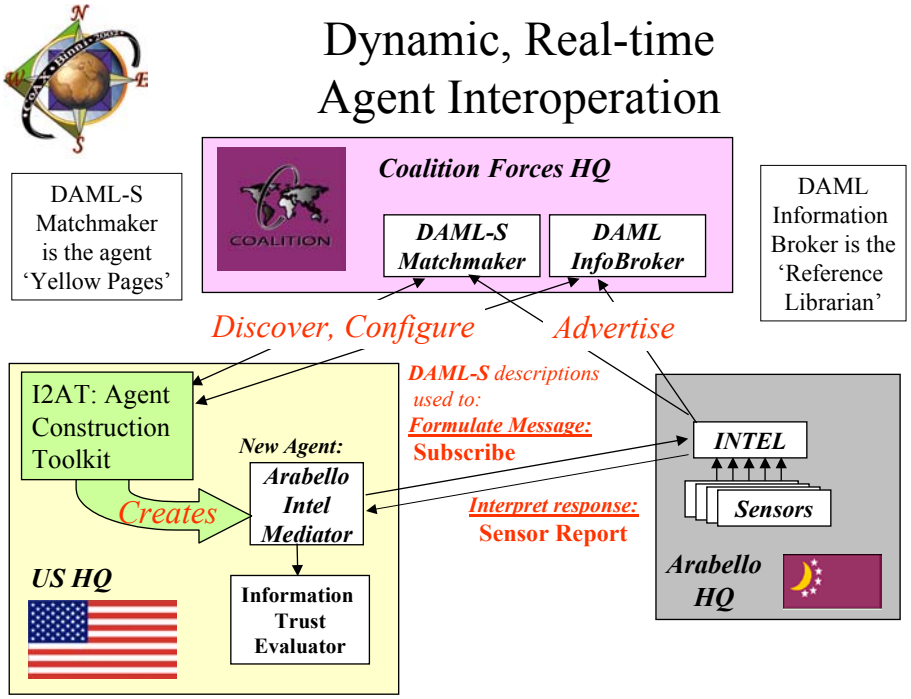


Figure 11: Agent Interoperation in the CoAX Binni Scenario

8.2.2 DARPA AGENT MARKUP LANGUAGE –SERVICES (DAML-S) DESCRIPTION

The first step in achieving this fast dynamic interoperability is to create a DAML-S description of the service an agent provides. A DAML-S description consists of three components. The *service profile* gives a high level description of the inputs, outputs, preconditions, and effects of the service. By semantically marking these properties with links to shared ontologies, matchmaking agents can connect client agents with appropriate service providing agents. The *service model* defines the sequence of agent messages that go back and forth. Finally, the *service grounding* tells the client agent exactly how to format the messages defined by the service model, and how to get them to the service agent. In the CoAX experiment, two information provider services from Arabello agents were made available to the coalition. The query service allowed a client agent to request a list of contact reports for a specific time period. The subscribe service allowed a client agent to be continually informed about all new contacts of a certain type. The next three sections will describe the DAML-S descriptions created for the Arabello agent providing these services in the CoAX experiment.

8.2.3 DAML-S SERVICE PROFILE

The service profile defines the inputs required from the client agents, and the outputs provided by the service agent. These inputs and outputs are linked to DAML ontologies, some created specifically for this

scenario, and others that were part of a preexisting DAML web ontology knowledge base[11]. Both the query and subscribe services required a region of interest as an input. Two classes of regions were defined, a *RectangularRegion*, which consisted of two points forming the corners of a rectangle, and a *PolygonRegion*, which consisted of a set of points defining the vertices of a polygon. These points were latitude, longitude, and elevation triples, members of a *LatLongCoordinate* DAML class (found in the DAML knowledge base). Some specific instances of these region classes were defined to represent some important regions to the scenario, such as a region encompassing the Red Sea. Both services also required an object classification, which defined what type of object the client was interested in reports about. A *Vehicles* ontology, based on the CYC transportation ontology, was created that defined a class hierarchy of the types of vehicles expected to be in a naval scenario. Finally, the query service also required a time interval, which was linked to a time ontology defined by Kestrel Institute and found in the web knowledge base.

The output from both services was linked to a *SensorReport* DAML class defined in an anti-submarine warfare ontology we created for this scenario. The *SensorReport* contained a number of properties including bearing, range, time, sensor position, and object classification. Some of these properties were linked to the same DAML classes used to define the inputs, such as the *LatLongCoordinate* and Kestrel Time classes. Others such as the bearing property were defined in a newly created measurement ontology.

8.2.4 DAML-S SERVICE MODEL

The service model for the query service was quite simple. Consisting of a single input message and corresponding output message, the query service model is just an *AtomicProcess*. The subscribe service model was slightly more complicated, since multiple output messages can be generated.

8.2.5 SERVICE GROUNDING

Since the DAML-S grounding tells the client agent exactly how to format its input messages and how to read, we created a grounding ontology for a CoABS Grid agent, and created two instances for the query and subscribe services. A *GridGrounding* consists of three main parts. First, in order to send a message to a Grid agent, the client needs to get a handle on the service provider's *AgentRep* object. Getting a specific *AgentRep* requires knowing the agent's unique *ServiceID*, or knowing enough fields of the agent's *CoABSAgentDescription* so that a search will return a unique *AgentRep*. The first part of our *GridGrounding* provides either a *ServiceID* or a *CoABSAgentDescription*. This allows the client agent parsing the service description to locate the *AgentRep* of the service and eventually send a message to it.

Once the client agent locates the service provider agent, it next needs to understand how to format its initial input message and how to understand the format of the expected response messages. This procedure is addressed by the next two sections of the *GridGrounding*. We defined an ontology of message types consisting of a hierarchy of classes from simple *StringMessages* through FIPA compliant messages and CoABS Grid Messages. An important member of the message ontology, which most of the more complex message classes are subclasses of, is the *KeywordValue* message. The *KeywordValue* message is composed of any number of *KeywordValuePairs* and a template string that defines how to format *KeywordValuePairs* into a single string. A *KeywordValuePair* consists of two properties, a string keyword, and a value that can be linked to any DAML class. Another type of message format is the *DAMLRDFString*. This corresponds to an instance of a DAML class defined in RDF syntax.

In order to complete the definition of a message format, there needs to be a link from the inputs or outputs defined in the service model to the properties of the message classes defined in the service

[11] http://www.daml.org/ontologies/

grounding. The client agent can use these links to determine where to put the input values, and then use the message ontology to derive the exact message string to be sent to the service.

For the Arabello information provider query and subscribe services, the input and output messages were defined to be a Grid Messages with a *FIPAMessage* as content. *FIPAMessage* consists of a string performative and several *KeywordValue* pairs, the most important of which is the "content" keyword. The content of the *FIPAMessage* was a *DAMLRDFString*. This means that the actual message content being sent back and forth was marked up by DAML tags. Semantically marking up the message content allowed the use of powerful content filtering techniques described in Section 8.2.

By using the information present in the service grounding, the process of constructing input messages and parsing the response output messages can be fully automated. The client agent needs only provide instances of the input parameters defined in the service model and then call a simple method to access the service and send its input message. The output messages provided by the service can be parsed using another method which pulls out the values of the output parameters specified in the service model from the received message and returns them to the client agent as part of a simple table.

8.2.6 DAML-S SERVICE DISCOVERY

In order to initiate this dynamic agent interaction, the client agent must first learn of the existence of the service agent, and then realize that this service agent can provide the needed data. Two separate methods to accomplish this were implemented, the first using the services of a DAML Semantic Matchmaker agent, and the second using the agent registration and search features of the CoABS Grid itself.

8.2.7 DAML-S MATCHMAKER

The DAML Semantic Matchmaker developed by CMU (Paolucci et al., 2002) is a direct solution to our service discovery problem. The service provider agent sends the matchmaker agent its DAML-S service profile. The client agent can then create a service profile template, describing what it's looking for in a service agent. For example, the coalition agent created a template defining the inputs it could provide, namely an instance of the DAML class Region corresponding a region encompassing the Red Sea, and a DAML class representing the concept of a submarine. The required outputs from the service were then added to the template, in this case a sensor report DAML class containing bearing and range properties. The Semantic Matchmaker then matches this template against existing service profiles that agents had registered, returning matching service profiles to the client agent.

8.2.8 GRID MATCHMAKING

In the event that the Semantic Matchmaker agent is not available, a simpler matchmaking service was implemented using the Grid. We created a Grid capability class called *DamlServiceDescription*, which contained links to the DAML-S description files (profile, model, and grounding). The *DamlServiceDescription* initialization code parsed the service profile portion and stored the data in the local data fields. This gave the Grid's predicate test search routines fast access to the service profile data without requiring parsing raw RDF. The client agent creates its service profile template and passes it to the Grid predicate test search routine. The Grid software then calls a special function to match the client's template service profile with the service profiles registered by service agents. The matching system that was implemented for this predicate test was a very simple one, requiring exact matches for the input and output properties. DAML relationships such as *sameClassAs* and *equivalentTo* were ignored. This search routine returned to the client agent the *DamlServiceDescription* objects found. The full DAML-S description of the service could then be accessed by following the URI links present in the data fields of the *DamlServiceDescription*.

8.2.9 CLIENT BEAN

With the DAML-S service profile, model, grounding, and the matchmaker services, dynamic interoperability becomes a much more easily realized feature of multi-agent systems, with the burden on the client agent greatly reduced. If the client agent is not directly using the ontologies referenced by the service agent, however, some human assisted translation is necessary. One solution to making the process easier for the client is to provide a graphical interface allowing a client agent operator to specify the appropriate inputs. If the client agent is using a Java Beans development environment, integrating this graphical interface becomes a trivial task. This was the case for the CoAX experiment, where Lockheed's Interoperable Agent Toolkit[12] acted as the client agent requesting data from the Arabello information provider.

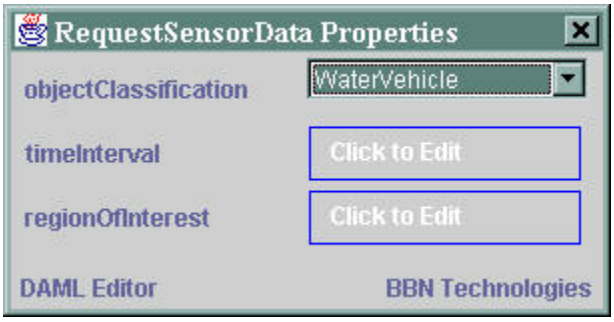


Figure 12: Query Service Input GUI

Figure 12 shows the main window of the query service input graphical interface. According to the service profile and model, the query service requires three inputs, a *Region*, a *TimeInterval*, and an *Vehicle* class.

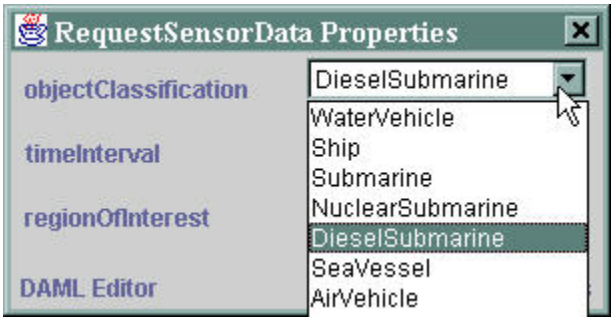


Figure 13: Object classification definition

The *Vehicle* class is simply a URI to a DAML class defined in the *Vehicle* ontology, and the possible values can be derived by simple parsing that ontology and retrieving all possible subclasses of *Vehicle*. Figure 13 shows the user selecting one of these possible *Vehicle* classes.

[12] http://www.atl.external.lmco.com/overview/programs /IS/I2AT.html

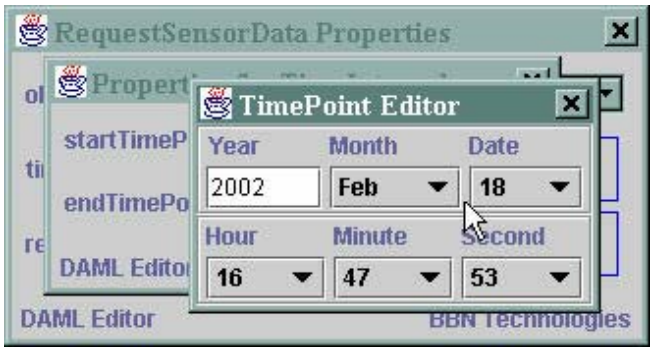


Figure 14: Time Interval GUI

Figure 14 shows the interface for specifying a time interval, which consists of two time point objects.

Figure 15: Region of interest GUI

The interface for defining the region of interest for the query service is shown in Figure 15. Three possible choices are given, the first two, Rectangular Region and Polygonal Region correspond to DAML classes defined in the measurement ontology. The Predefined Region choice allows the user to select an instance of a Rectangular or Polygonal Region already defined, such as the Red Sea region. Figure 16 shows the interface used for specifying the exact latitude, longitude, and elevation points comprising the region.

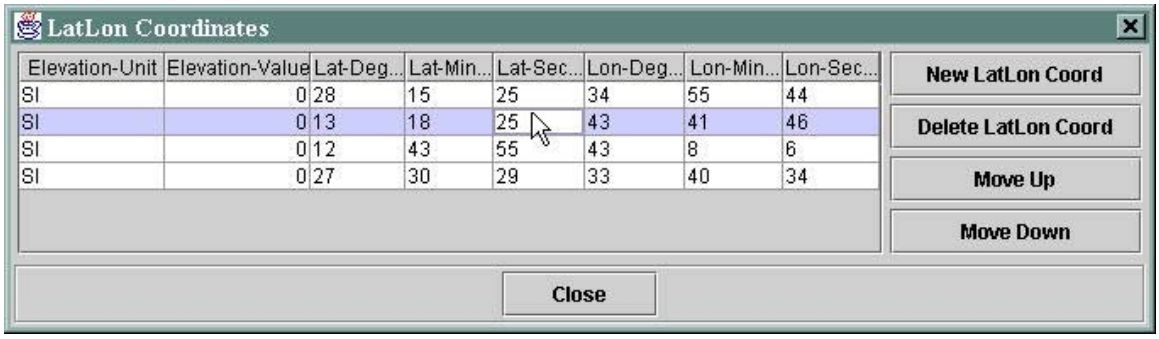


Figure 16: The Red Sea predefined region

8.3 SEMANTIC FILTERS

In some situations, restricting agent communication based solely on attributes of the sending or receiving agent may be a too coarse grained form of control. Often, the content of the communication contains the critical features for determining if and how to filter the communication. For example, one might wish to restrict the sharing of sensitive personal data or proprietary business information. Just as we have used DAML-S as the language to express the capabilities and services of agents, we use DAML to express the semantic content of the information exchanged through services. Using DAML, we present a system for specifying and enforcing a semantic content filter. This system requires no modification of source code, allowing content filters to be dynamically defined during run-time.

We have demonstrated this system as part of the CoAX technology integration experiment (See Figure 17). In the CoAX scenario, the country of Arabello joins the coalition. Consequently, a number of new agents and agent services need to be dynamically made available to coalition agents. These agents and services are dynamically discovered, resulting in a number of new agent interactions. For example, one interaction involves a coalition agent tasked to locate a hostile submarine and an Arabello agent capable of providing sensor reports from an underwater sensor grid. As new coalition partners, Arabello system administrators dynamically allow sensor contact reports to be sent to the coalition agent, but for security reasons, restrict the range of messages that could be sent outside of the Arabello domain. The limitation, described as part of a policy represented in DAML, limits these outgoing messages to those whose content are reports about a specific class of submarine, belonging to the enemy forces, but disallowing reports on other ships, such as those of Arabello itself.

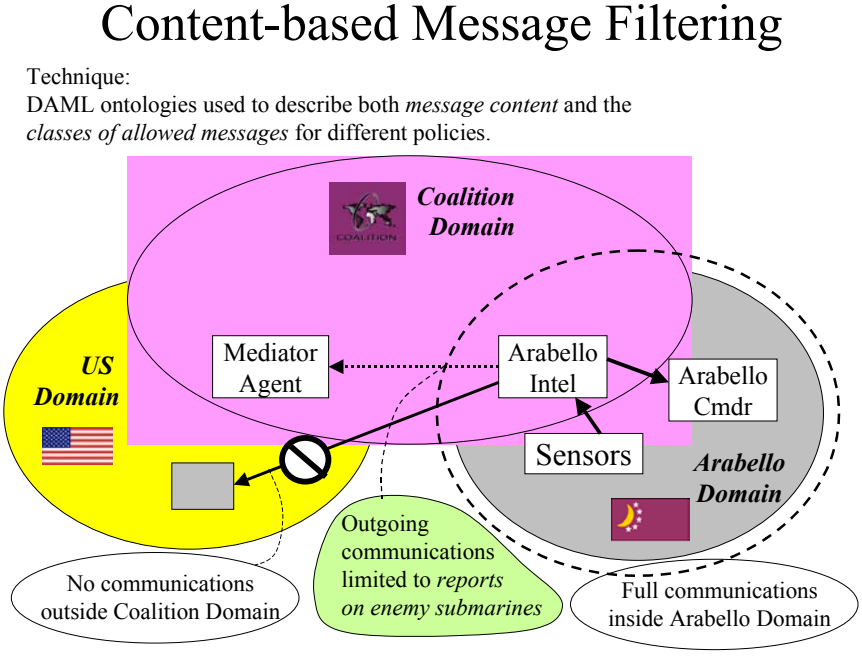


Figure 17: Semantic Filtering in the CoAX TIE

8.3.1 FILTER SPECIFICATION

In order to enable domain administrators to specify a message content filter, it first becomes necessary to access the DAML ontology describing the possible outgoing message contents. This information is available as part of the DAML-S service description for each agent: specifically, the output message property in the agent's service process model. A DAML-S process model describes, among other things, the DAML classes representing the types of each service's inputs and outputs. The approach to specification of message filters we adopted enables a KAoS domain system administrator to specify or define a subclass of the most general allowed class of input or output messages that will be permitted to be sent or received by some class of agents.

For each content filter, a GUI is dynamically generated that lets the system administrator build up a specialized class definition that will be used as a filter by comparing it to each message being sent between the classes of agents covered by the policy. If the message is subsumed by this class, then the message is permitted to be sent or received in the case of a positive authorization policy, or blocked in the case of a negative authorization policy. The specialized DAML class is created by placing additional property range restrictions on the properties of the DAML class describing the message content specified in general by the agent's service process model. For each property to be restricted, we can create two types of DAML restrictions. The first, called a *toClass* restriction, requires that the value of the property be a member of a certain DAML class, the second, called a *hasValue* restriction, requires the property to have a specific value.

KPAT, our policy administration tool, provides an interface by which a system administrator can specify policies for interactions based on the properties of services and agents discovered dynamically at run-time. For the content filtering policies, the properties of the DAML class representing the output of each service are shown to the administrator in a dynamically generated GUI, along with a *toClass* restriction editor, and a *hasValue* restriction editor. The *toClass* editor contains a list of previously defined classes that could be selected in order to further restrict the property's range. This list is developed by expanding and linearizing the subclass tree of the DAML class defining the original

daml:range of the property for the message class specified for the service. The *hasValue* editor allows the entry of freeform text representing the value of the property, or selecting from known instances of the range class, if it consists of a pre-defined closed list. Special graphical editors for certain DAML classes (e.g., dates and latitude/longitude coordinates) are also provided. In addition, if the daml:range is a daml:Class, meaning that the range of values of the property is a set of DAML class URIs, the possible values are automatically generated, just as for the *toClass* restriction.

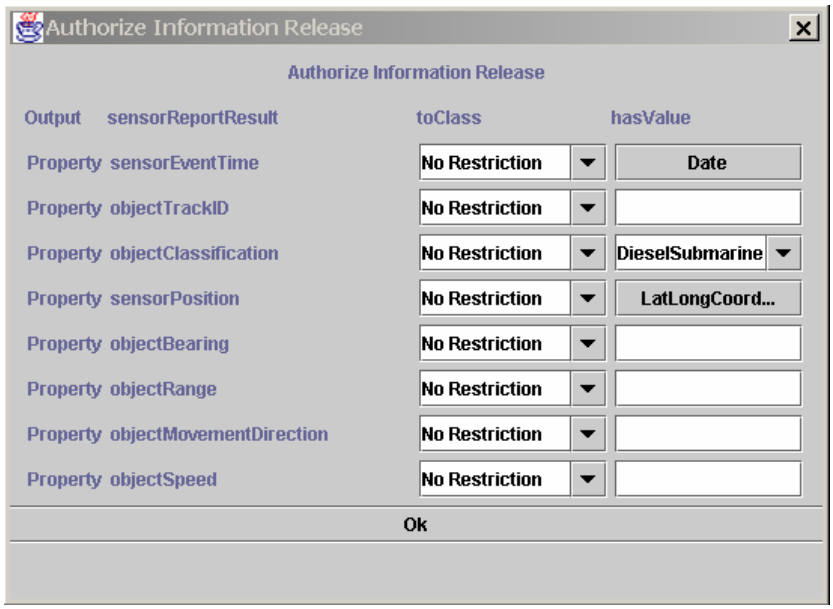


Figure 18: Semantic Content Filter Specification Dialog

The interface generated for our CoAX example is shown in Figure 18. By selecting 'DieselSubmarine' in the *hasValue* editor for the property *objectClassification*, the system administrator is restricting outgoing *sensorReportResult* messages from the Arabello agent that is the subject of the policy to those with property *objectClassification* having the value 'DieselSubmarine'.

8.3.2 FILTER GENERATION

Once the classes describing the message properties that indicate which messages are to be filtered is specified through this GUI, a persistent DAML class is created, representing the full set of these restrictions. This new DAML class is defined as an intersection of the original output message class and a class expression (specified using daml:Restriction) for each property that is further restricted by a *toClass* or *hasValue* expression. In our example, the resulting DAML class is given below:

```
<daml:Class rdf:ID="RestrictedASWSensorReport">
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="&asw;#ASWContactReport"/>
    <daml:Restriction rdf:ID>
      <daml:onProperty rdf:resource="&asw;#objectClassification"/>
      <daml:toClass rdf:resource="&vehicles;#DieselSubmarine"/>
    </daml:Restriction>
  </daml:intersectionOf>
</daml:Class>
```

8.3.3 POLICY CONFLICT RESOLUTION

Changes or additions to policies in force, or a change in status of an actor (e.g., an agent joining a new domain or moving to a new host) or some other entity require logical inference to determine first of all which policies are in conflict and second how to resolve these conflicts. We have implemented a general-purpose algorithm within KAoS for policy conflict detection and harmonization whose initial results promise a high degree of efficiency and scalability.

When administrators commit mutually inconsistent policies, KAoS first checks to see whether one of the policies takes precedence over the other. The higher priority policy remains in force unchanged while the lower priority policy becomes the subject for policy harmonization. The result is zero, one, or several harmonized policies. Following harmonization, the user is notified and given an opportunity to resolve any remaining issues and approve the results of policy conflict resolution. Following user approval, any obsolete policies are removed and new policies are sent to the appropriate enforcers. Details on policy conflict resolution may be found in (Suri et al., Submitted).

8.3.4 POLICY ENFORCEMENT

In order for the filter to be enforced, the newly generated DAML class is provided, through the KAoS policy framework, to a policy enforcer. A message content policy enforcer uses a message content policy guard, to test whether the policy applies to each message. This guard was developed using the Java Theorem Prover (JTP) developed at Stanford KSL[13]. The enforcer provides to the guard the class describing the filter for which the policy is defined along with the URIs of the DAML ontologies referenced by this filtering class. Subsequently, whenever messages being transmitted between the classes of agents covered by the policy are detected, the content of those messages (also represented in DAML) is given to guard for comparison to the message filter class. This test succeeds if the message content is inferred to be an instance of the filter class. If it is, and the policy is a positive authorization policy, then the message passes the filter, and the enforcer permits it to be sent to its destination. If the policy is a negative authorization policy and the test succeeds, then the message is blocked. The reasoning provided by JTP in conjunction with a set of axioms defining the semantics of the DAML language and the sets of ontologies referenced by the message filter class and the message being tested enables the necessary reasoning about *toClass* and *hasValue* restrictions of the policy.

In order for our semantic content filters to be tested against agent messages, the message content must be a DAML instance. For the CoAX demonstration, all agent messages were specified directly in DAML, by using DAML-S in conjunction with a grounding mechanism that wrapped the DAML message content (a string in RDF syntax) in a CoABS Grid message. If, however, the content of the message was in some other form, a mapping would need to be defined between the raw content of the message and a semantic encoding as a DAML description in order to use this approach to message filtering.

9. CONCLUSIONS

The CoABS program has provided the opportunity to understand what the capabilities of software agents can be, and how to make effective use of large numbers of agents in the accomplishment of specific, large-scale tasks. We focused on a number of areas critical for the tasking and managing of teams of agents and humans. We have developed techniques for ‘agentizing’ existing systems in order to more quickly integrate these systems into larger, human managed organizations. We created mechanisms for dynamically forming, tasking, and monitoring teams of agents in support of human needs and objectives. We have developed a framework for dynamic information sharing among teams of humans and agents working as parts of larger, frequently hierarchical, organizations. We have developed

[13] http://www.ksl.stanford.edu/software/JTP

techniques for automatically translating information across differing ontologies thereby allowing agents to 'speak the same language'. We have developed DAML-S services descriptions in order to express the capabilities and services of agents, as well as techniques for monitoring and controlling agent communication based on the semantic content of the communication messages, which enable agents to communicate interoperability.

10. ACKNOWLEDGMENTS

We gratefully acknowledge all those who contributed to the CoAX project, whose names are too numerous to mention. We thank Prof. James Hendler, DARPA Program Manager for CoABS for organizing the MIATA group and giving us tremendous latitude in developing the project. We are also grateful for the contributions of the MIATA working group, including Karen Myers and David Moreley (SRI), George Ferguson and James Allen (University of Rochester), Sebastian Thrun and Jaime Schulte (CMU), Brian Drabble and Najam-ul Haq (CIRL, University of Oregon), Drew McDermott (Yale University), Doug Smith and Stephen Westfold (Kestrel Institute), and Steve Ford (OBJS, Inc.).

We also thank LCDR Schmorrow, Prof. Hendler's successor, for his support of our efforts to integrate the results of MIATA into CoAX, and all of those who participated with us in that effort.

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

11. REFERENCES

Ankolekar, Burstein, Hobbs, Lassila, Martin, McIlraith, Narayanan, Paolucci, Payne, Sycara, & Zeng. (2001) DAML-S: Semantic Markup for Web Services. To appear in Proceedings of the 2001 Semantic Web Workshop, Stanford, CA.

Arnold, K., O'Sullivan, B., Schiefler, R. W., Waldo, J. & Wollrath, A. (1999). The Jini Specification. The Jini Technology Series, Addison-Wesley:Reading, MA.

Bradshaw. J. M., Dutfield, S., Benoit, P., and Woolley, J. D. (1997). KAoS: Toward an Industrial-Strength Open Agent Architecture. In J. Bradshaw (Ed.), Software Agents. Cambridge, MA: MIT Press.

Bradshaw, J. Uszok, A, Jeffers, R., Suri, N., Hayes, P., Burstein, M., Acquisti, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Johnson, M., Kilkami, S., Lott, J., Sierhuis, M., & Van Hoof, R. (Submitted) Representation and Reasoning for DAML-Based Policy and Domain Services in KAoS and Nomads. AAMAS 2003, July 2003, Melbourne, Australia.

Burstein, M. H. & Diller, D. E. (in-press) A Framework for Dynamic Information Flow in Mixed-Initiative Human/Agent Organizations to appear in Applied Intelligence.

Burstein, M. H. & Diller, D. E. (2001). Mixed-Initiative Agent Team Formation, Tasking and Failure Handling. Proceedings of the AAAI Fall Symposium on Intent Inference for Collaborative Tasks, Falmouth, MA.

Burstein, M. H. & Emerson, T. (1999). Development of a Constraint-based Airlift Scheduler by Program Synthesis from Formal Specifications, Proceedings of the 1999 Conference on Automated Software Engineering, Orlando, FL, September, 1999.

Burstein, M. H., Ferguson, G., & Allen, J. (2000) Integrating Agent-based Mixed-initiative Control with an Existing Multi-agent Planning System. In Proceedings of 2000 International Conference on Multi-agent Systems (ICMAS).

Burstein, M. H., & McDermott, D. (1996). Issues in the Development of Human-Computer Mixed-Initiative Planning. Cognitive Technology, B. Gorayska and J.L. Mey (Eds.), Elsevier, Jan, pp. 285-303.

Burstein, M., McDermott, D., Smith, D, & Westfold, S. (2000a) Derivation of Glue Code for Agent Interoperation. In Proceedings of Autonomous Agents 2000, Barcelona, Spain.

Burstein, M., McDermott, D., Smith, D., & Westfold, S. (2000b) Formal Derivation of Agent Interoperation Code. In Proceedings of the 2000 Workshop on Formal Approaches to Agent-based Systems, Washington, DC.

Burstein, M. H., Mulvehill, A. M. & Deutsch, S. (1999a) Mixed-Initiative Tasking and Management of Software Agent Teams. Proceedings of AAAI Workshop on Mixed-Initiative Intelligence, Orlando, FL.

Burstein, M. H., & Mulvehill, A. M., & Deutsch, S. (1999b). An Approach to Mixed-Initiative Management of Heterogeneous Software Agent Teams. Proceedings of Hawaii International Conference on Systems Sciences.

Cohen, P. R., Levesque, H. R., & Smith, I. (1997). On Team Formation. In J. Hintikka, and R. Tuomela (Eds.), Contemporary Action Theory. Synthese.

The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara). DAML-S: Web Service Description for the Semantic Web. (2002). The First International Semantic Web Conference (ISWC).

Deutsch, S. E. (1998). Interdisciplinary foundations for multiple-task human performance modeling in OMAR. In Proceedings of the 20th Annual Meeting of the Cognitive Science Society, Madison, WI.

Deutsch, S. E., & Adams, M. J. (1995). The operator-model architecture and its psychological framework. 6th IFAC Symposium on Man-Machine Systems. MIT, Cambridge, MA.

Ferguson, G. and Allen, J. (1998). TRIPS: An Integrated Intelligent Problem-Solving Assistant. In Proceedings of AAAI-98. AAAI Press.

Finin, T., Labrou, Y., & Mayfield, J. (1997). Kqml as an agent communication language. In J. Bradshaw, editor, Software Agents. AAAI Press/MIT Press.

Firby, R. J. (1994). Task networks for controlling continuous processes, in Proc. Int. Conf. AI Planning Systems, Menlo Park, CA.

Florescu, D., Raschid, L., & Valduriez, P. (1996). A methodology for query reformulation in C is using semantic knowledge. Int. J. of Cooperative Information Systems.

Groff, J. R., & Weinberg, P. N. (1998). The Complete Reference SQL. McGraw-Hill.

Grosz, B. J. & Kraus, S. (1996). Collaborative plans for complex group action, Artif. Intell., 86, 269-357.

Grosz, B. J. & Kraus, S. (1999). The evolution of SharedPlans, in M. Wooldridge and A. Rao, Foundations and Theories of Rational Agency, 227-262.

Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., & Crespo, A. (1997). Extracting semistructured information from the web. In Proc. Workshop on Management of Semistructured Data, Tucson, Arizona.

Kiczales, G. (1996). Aspect-oriented programming. ACM Computing Surveys, 28(4es):154.

Martin, D. L., Cheyer, A. J. & Moran, D. B. (1999). The open agent architecture: A framework for building distributed software systems. Applied Artificial Intelligence, 13, 91-128.

McDermott, D., Burstein, M. & Smith, D. (2001) Ontology Matching in Transactions with Self-Describing Agents. To appear in Proceedings of the 2001 Semantic Web Workshop, Stanford, CA.

Menzies, T. (1999). Cost benefits of ontologies. Intelligence, 10, 27-32.

Milo, T. & Zokar, S.(1998). Using schema matching to simplify heterogenous data translation. In Proc. Conf. on Very Large Data Bases, 122-133.

Mulvehill, A. & Caroli, J. (1999). JADE: A tool for rapid crisis action planning, Proc. of the 1999 Command and Control Research and Technology Symposium, US Naval War College, Newport, RI.

Myers, K. L. (1993) "User's guide for the Procedural Reasoning System," Artificial Intelligence Center, SRI International, Menlo Park, CA., 1993.

Myers, K. L. (1996) Advisable Planning Systems. In A. Tate (Ed.), Advanced Planning Technology. Menlo Park, CA: AAAI Press.

Myers, K. L., & Berry, P. M. (1999). At the boundary of workflow and AI, in Proc. AAAI 1999 Workshop on Agent-Based Systems in The Business Context.

Paolucci, M., Kawamura, T., Payne, T. R., Sycara, K.(2002). Semantic Matching of Web Services Capabilities. In Proceedings of the 1st International Semantic Web Conference.

Papakonstantinou, Y., Gupta, A., Garcia-Molina, H., & Ullman, J. (1995). A query translation scheme for rapid implementation of wrappers. In Proc. DOOD'95.

Rathmell, R.A. (1999) A Coalition Force Scenario 'Binni - Gateway to the Golden Bowl of Africa', in Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces, (ed. Tate, A.), 115-125, Edinburgh, Scotland, 10th-11th May 1999.

Rich, C. & Sidner, C. (1997). COLLAGEN: When Agents Collaborate with People. In Proceedings of the International Conference on Autonomous Agents (Agents '97).

Sycara, K., Klusch, M., Widoff, S. & Lu, J. (1999). Dynamic service matchmaking among agents in open information environments. J. ACM SIGMOD Record, 28(1), 47-53.

Smith, D. R. (1990). Kids: A semi-automated program development system. IEE Transactions on Software Engineering 16.

Srinivas, Y. V. & Jullig, R. (1995). Specware: Formal support for composing software. In Proceedings of the Conference on Mathematics of Program Construction, B. Moeller, Ed. LNCS 947, Springer-Verlag, Berlin, 399-422.

Suri, N., Bradshaw, J., Burstein, M., Uszok, A., Benyo, B., Breedy, M., Carvalho, M., Diller, D., Groth, P., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J. (Submitted) DAML-based Policy Enforcement for Semantic Data Transformation and Filtering in Multi-agent Systems. AAMAS 2003, July 2003, Melbourne, Australia.

Tambe, M. (1997). Towards flexible teamwork, Journal of Artificial Intelligence Research, 7, 83-124.

Tambe, M., & Zhang, W. (2000). Towards Flexible Teamwork in Persistent Teams: Extended Report. Autonomous Agents and Multi-agent Systems, 3, 159-183.

Winograd, T. & Flores, F. (1987). Understanding Computers and Cognition: A New Foundation for Design, Addison-Wesley:Reading, MA.

12. APPENDIX A - COAX DEMONSTRATION SCRIPT

Arabello Startup:

1) Run Grid HTTP when told to
2) Run Arabello Agents when told to, wait roughly 1 minute
3) Run Commander GUI
4) Run KaoS Servlet Runner
5) Run KPAT

6) Minimize everything except for KPAT gui.

Step 56: Coalition Starter Pack is received

[First, a set of Arabello agent domains has to be created. This enables Arabello to partition their information and to accommodate their national sensitivities and security requirements. The Binni-Coalition SysAdmin creates the overall Arabello-HQ domain.]

[Now that the Arabello-HQ Domain has been added to the coalition, the Arabello-HQ SysAdmin can add subdomains for the agents that will be part of the coalition contingent and for the private agents that represent the sensors on their ships.]

1) Select the “Namespaces” tab in the left hand panel
2) Press the “Load Namespace” button at the bottom of that panel
3) Select the “ArabelloDomain” ontology from the file list
4) Press the “Load Selected Namespaces” button

[wait roughly 20 seconds]

4a) Start the IX Process Panel by double clicking on the Icon

[Now the Arabello domains can be populated with agents which register [STAGE- 1: BBN click on "Register Agents" on GUI - don't have to show this. NB: If we make this look too complicated people say "If I have to do this with 1000s of agents - no thank you!"] with the respective KAoS Domain Managers [STAGE- 2: [LT-09] => [PROJ-01] Show Anaconda - with the domain and agents appearing and beginning to interact as each is created in KPAT]]

[Anaconda should be showing here]

5a) Bring up the Commander GUI (Java Icon with no title)

5b) Select the Tools/KaoS Domain Registration menu item

5) Click on “Register Agents in KaoS Domains” button.

[Wait roughly 20 seconds]

6) Click Refresh on the KPAT gui

[Wait roughly 10 seconds]

[Coalition and Arabello SysAds now use the Policy Administration Tool [STAGE- 3: Show KPAT being used to create example policy - that enables the Arabello-Intel agent to communicate and do some filtering] to set up the agent policies (like SOPs and Standing Orders) for Coalition Communication with Arabello domains - so that all the agents behave as required - and Arabello does likewise. These policies enact behaviour changes in the agents which will be effected at run-time, without either knowing anything about the agent code or having to change any of it in any way]

6.1) Select the Actor Classes tab in the KPAT Gui

6.2) Select "Members of the Domain Arabello-HQ"

6.3) In the right pane, click on the Load Button

6.4) Select CoAXPolicy1.msg

6.5) Click Open

6.6) Click Commit

[Wait 20 seconds]

6.7) Click OK

[Arabello creates a policy restricting their Intel Agent to only provide INTEL reports on the class of submarines used by Agadez (Diesel). This is an example of restricting communications by content, rather than just by the domain of the sender and receiver]

7) Select Arabello-Intel by navigating the agent tree in KPAT (4 mouse clicks)

8) On the right panel, select DAML from the pull down menu

9) Click on ADD

10) A Create Daml Dialog box pops up

[Domain and policy information is represented using DAML - the DARPA Agent Markup Language developed jointly with the WorldWide Web Consortium - and combined with KAoS components to provide powerful reasoning, policy deconfliction, and policy enforcement capabilities. Policies prevent any Arabello agents from communicating outside their country's domains except those that belong to the coalition contingent]

17) Type in "BLOCK-CONTENT" in the name field

18) Type "2" into the Policy Priority field

19) Select CommunicationAction from the available action list

20) Click on the Select button

21) Click on the Lock Selection button

22) Select carriesCoaxMessage from the "Available Roles" list

23) Click on the Select button next to the "Available Roles" list

[AuthorizationDialog pops up, this can take a few seconds]

24) In the Authorization Dialog, select DieselSubmarine from the objectClassification list box in the toClass column.

25) Click OK

[Back to the CreateDamlContent dialog]

26) Choose hasDestination from the roles list

27) Choose the Select button

28) Choose Class button

29) Select “Members of DomainBinni-Coalition” from the “Available Ranges” list

30) Click on Select

31) Click on “Add Target”

32) Click on OK

[Dialog box closes, back to main KPAT screen, you can see the policy listed]

33) Click “Commit”

[wait 20 seconds, but we’re done here and can move on to the next step]

[Coalition contingent policies allow Arabello to receive any incoming messages from the coalition, while in turn allowing only certain kinds of Arabello sensor information to be given out to the coalition. Automated policy conflict detection and resolution is displayed graphically so that the SysAds can respond it the conflicts. In addition, we can see in Anaconda how certain kinds of messages from Arabello sensors pass through while others are blocked]

[Coalition tasks Arabello to provide sensor data]

33) Receive a process panel to “provide intel”

34) Choose Tools/Provide Services on the Commander’s GUI (TAICHI)

35) Double click on services

36) Click on Intel Agent

37) Check the boxes under Binni-Coalition for sensorQuery and sensorSubscribe

38) Click OK

39) Back in the process panel, mark the request as done by selecting done in the right hand side list box.

[MAD-Client]

Step 83

[Coalition decides to provide a feed to Arabello from the MAD sensor on the Australian Ship which is now operational again]

Double click on the Mad Client icon

[wait 5 seconds]

Type in “MAD-Data-Source” in the text box

Click on “get images”

[Mixed Initiative tasking]

Step 88

[Arabello is tasked by CF HQ to provide the ‘bottom of the box’ by deploying ships with ASW capability to monitor the Red Sea below Latitude 17N. The Arabello commander uses his mixed-initiative interface to generate a plan and task three ships to patrol this area]

1. On Commader’s GUI: Select the Script button
2. A Dialog box pops up, select the Edit button

3. A Dialog box pops up, Select Region Sweep
4. On the Map gui, select the draw region button
5. Draw a rectangle (4 mouse clicks)
6. On the edit dialog box, select “Choose Region”
7. On the edit box, select “Choose Assets”
8. On the asset table, click on the first ship, then shift-click on the last ship.
9. On the edit dialog box, select ok

[Ships start to move, ship plan will show up as arrows on the map gui]