An Approach to Mixed-Initiative Management of Heterogeneous Software Agent Teams

Mark H. Burstein, Alice M. Mulvehill, Stephen Deutsch
(burstein@bbn.com, amm@bbn.com, sdeutsch@bbn.com)
BBN Technologies (a unit of GTE Internetworking)
10 Moulton St., Cambridge, MA 02138

Abstract

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. We believe that practical multi-agent systems developed will often be comprised of small teams of heterogeneous agents, under direct supervision by users acting as "team leaders". We are now developing an environment for investigating approaches to controlling small to medium-sized groups of agents as coordinated teams. This environment will be used to explore mixed-initiative approaches to planning for the activities of agent teams, and managing them during execution. Our approach arises out of a long-standing interest in mixed-initiative planning systems [5]. In this paper, we discuss our approach to mixed-initiative agent team management, some representational issues involved in identifying compatible agent team members and the capabilities needed to monitor team execution.

1. Introduction

We are now in an era of fast-paced growth in the development of agent-based systems technologies. As we seek to make effective use of heterogeneous agents in tasks requiring some degree of inter-agent coordination, we must develop strategies and mechanisms for tasking and managing them during execution. Unfortunately, the functionalities, capabilities, and limitations of agents are evolving rapidly, and many of the distributed software components that are likely to need in real-world systems are not agents at all by most current definitions. To develop a realistic strategy for user-controlled planning and management of multi-agent systems, we must expect that a range of "agent" and "less than agent" software components will be involved, and begin to characterize their functional capabilities, domain knowledge, information requirements, and decision authority, in order to properly task them.

The objective is the design and development of mixed-initiative tools that support humans in planning, tasking and managing software agent teams. Our approach is to simultaneously build mixed-initiative user support agents and an environment in which we can experimentally explore approaches to user interaction with these team building and managing agents. Our initial domain is military logistics planning and scheduling.

1.1. Approach

To effectively support the mixed-initiative management of agent teams in this kind of planning task, we must address support for such critical areas as:

(1) HCI techniques for interactively defining tasks and specifying roles and communications between agents,
(2) Mixed-initiative support agents to help users identify appropriate agents for tasks, and ascertain that agents the can communicate compatibly, satisfy timing dependencies, and provide structural support for execution status monitoring,
(3) Support for the collection, summarization and visualization of agent task status, to convey to users the impact of problems on major team objectives,
(4) Support for dynamic retasking to surmount execution-time problems, changing conditions and objectives.

Our approach to team formation and task management is organized around mixed-initiative tools for plan construction, visualization and maintenance of plans and associated agent taskings. As we are working primarily in a domain where the products are themselves plans, it is appropriate to provide a common ontology and gestural vocabulary for specification of major domain objectives and the tasks to be done by agents supporting those plans. Our aim is to facilitate user interaction by direct manipulation [21] of domain-specific visualizations of task representations, and avoid (for pragmatic reasons) issues of natural language-based interaction. We do, however anticipate that multi-modal dialog management will be critical for handling issues such as maintaining the locus of joint planning activity, delegating and retracting change authority, and maintaining dialog context (as when switching from planning to execution management or evaluation).

0-7695-0001-3/99 $10.00 (c) 1999 IEEE

The mixed-initiative aspects of the system we are constructing will arise from the behavior of several classes of user support agents. **Planning support agents** will "watch" the manipulations of users developing plans (primarily graphically) and suggest appropriate ways to refine the specified tasks, identify appropriate **functional agents** to fill team roles addressing the specified tasks, and identify functional and communication gaps that remain given the choices made for those assignments. These activities will result in short dialogs with users, providing suggestions and warnings, but leaving users with the final change authority.

This interactive process between users (the team leaders) and agents will also identify task information requirements, establishing the need for **information seeking and reformulation agents** to gather the information needed and provide it where and how it is required. Finally, the process must introduce as team members **monitoring agents** to assist in the collection and tracking of team performance and status, and report to users on issues and progress as needed.

We are taking a decidedly experimental approach to developing mixed-initiative controls for agent teams. Our initial suite of planning tools will be integrated into a prototype testbed with an **agent simulation environment** that enables us to explore mixed-initiative techniques for tasking agents without necessarily having a full complement of real agents. The testbed will enable us to experiment with different approaches to user interaction for managing teams and explore some general agent coordination issues, such as the identification of effective organizational control strategies for different classes of tasks.

We are fortunate to have available a simulation tool for agents with explicit goal and process representations called OMAR (Operator Modeling Architecture) [7, 9] that is well suited for modeling current and future classes of software agents and as a platform for these experiments. OMAR has been successfully used to model human performance in a variety of complex activities (cockpit modeling, underwater vehicle control, etc.). OMAR also directly supports the control of live, remote agents, so that we can extend our experiments to look at control of true heterogeneous software agents as well as simulating them.

1.2. An Example

As a hypothetical example, consider the logistics domain objective to move a quantity of food and other humanitarian supplies to an area hit by an earthquake or other disaster. A planning support agent assisting a military logistics user to plan this deployment might initially suggest tasking information agents to check the weather and airport conditions near the affected area, and propose or initiate other requests to find the closest supplies and aircraft to move those supplies. While those information requests were processed, agent taskings could be developed that would apply the results of those information queries to the generation of airlift flight plans. The resulting agent tasking plan might include communications with a variety of domain agent "specialists", such as flight route planners, aircraft cargo packing specialists, etc.

Now suppose the local weather agent could not supply requested information because one of its sources was unavailable due to communications disruptions. At the *expected completion time* of the weather information task (or before, if a status monitor anticipated the problem), a monitoring agent would bring this problem to the attention of the team leader. This alert would be accompanied by a warning that the tasks depending on this information (here, the flight route planner) still lacked needed inputs (*e.g.*, wind speed and direction). The team leader and support agents would then be responsible for tasking another agent to supply the missing information, perhaps by delegating it through a remote information mediator to a standard seasonal wind model. The planning support agent would then need to determine whether the information retrieved from that model required reformulation before passing it to the route planner, and , if necessary, direct a translation agent to translate and send the result to the route planner.

1.3. Monitoring Task Planning and Execution

Key points illustrated by this example are that task planning and execution are almost always interleaved, and that anticipation of future tasks (delegated or not) drives the near-term tasking of agents. Because task generation and delegation are interleaved, we need to consider them as part of the same mixed-initiative planning process, and expect dialog with the user to bounce between these two levels of decision making. In essence, agent task delegation should be viewed essentially as part of the same interactive process as the domain-specific activities of the system.

Many people (*e.g.*, Donald Norman [17]) have pointed out that human users of distributed systems of agents are unlikely to develop trust in their agent-based systems' capabilities until they can be employed, much like human subordinates, in a way that they can be carefully monitored and managed. Only when the agents can respond effectively to tasks, collaborate with other agents in the process, and appropriately handle a variety of failure conditions will people give them much autonomy. Ultimately this might imply a level of learning and adaptivity we don't see in most current-day agents, but in

the near term, it means that users must be able to develop appropriate expectations of agent capabilities, and be able to monitor agents that are tasked to see that they fulfill those expectations. For effective monitoring, users will need on-demand visibility into the activities of all tasked agents, be able to track and visualize their progress and receive feedback on problems and their implications so that they can be addressed as they occur. Given the amount of information likely to be involved in each agent's actions, effective summarization and visualization of agent activities will be critical.

2. Mixed Human/Agent Organizations

Existing architectures for agent-based software systems can be grouped into two classes:

- Loosely coupled systems of agents (homogeneous systems) where each agent typically plays a similar role. Agents may compete for resources or tasks using an economic model or other simple control structure.
- Tightly coupled systems of agents depending on cooperative behavior among heterogeneous classes of agents. These systems require adaptive agents or well defined communications and interaction patterns.

In fact, most large organizations have mixtures of these two kinds of structure, depending on the tasks to be performed from a functional perspective. At higher levels of the organization, team-like organizations may predominate, with subordinate organizations or agents broken out by functional areas, or by geographical considerations. We see software agents for large organizations being organized similarly, so that they fit into the structures of the human organizations. In many ways, software agents will "sit beside" human agents in these organizational structures, and need to be tasked and managed similarly. Figure 1 shows our concept of how agent teams might be organized to fit into human organizations.

In the figure, human team leaders are supported by "personal assistant" agents to help with agent planning and monitoring for the team. To create a team to address some problem, a team leader first specifies high level task objectives, and begins to refine these down to the level of tasks that specific agents from the available pool can address. As plans for an objective are developed, and early information needs are identified, the team leader or an assistant may begin tasking local or remote agents to collect that information while planning for the rest of the team's activities continues. Remote agents and other teams may be tasked through mediators (*e.g.*, for database queries), or by direct messages (*e.g.*, to other human team leaders).

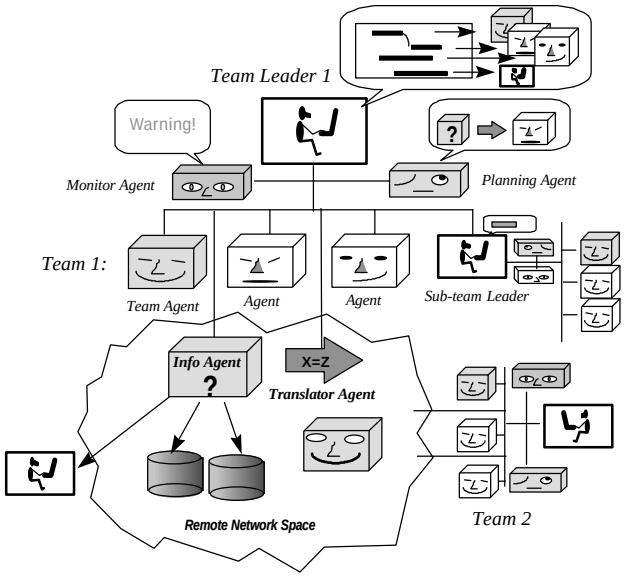

Figure 1: Agent Teams in a distributed organization.

2.1. What is an agent team?

An interesting question is what makes a set of software agents and other components into a team? Teams of people have been distinguished from *groups* [18] based on characteristics such as the presence of highly differentiated roles, interdependence among members, and the performance of tasks that require coordination among multiple individuals. Groups, in contrast, have homogeneous and interchangeable (non-specialized) members; the members often work independently (coordination is not required), and they perform tasks that could have been done, although perhaps not as well, by one person. Examples of groups falling under this definition include juries, panels of judges, and the ad hoc problem-solving groups in social psychology experiments on group decision making. Salas [20] similarly define a team as composed of multiple individuals, engaged in dynamic, interdependent interaction, pursuing a common goal or objective. Each team member has specific tasks or functions, and the tasks of different agents requires the dynamic interchange of information, coordination, and adjustment to task demands.

By these definitions, it seems clear that some types of multi-person or multi-agent tasks are best suited to teams, while others are best performed by groups, and still others require a hybrid of the two. We intend to explore tasks for sets of agents where each of these organizational forms is appropriate. Our main emphasis is on of heterogeneous agents with task differentiated roles, and well defined coordination requirements arising from the task plans.

Malone [14, 15] proposed that (human) organizational structures were driven by the kinds of coordination required, and that these in turn were driven by constraints based on shared resources, information and materials flow paths, etc. What he didn't discuss was the capabilities and limitations of the agents in a team or organization as a factor. These are critical factors in designing software agent organizations**,** as in real human ones. Organizational roles on a team are often based in part on the capabilities of the agents available to fill roles in a plan. In human organizations, this is often done adaptively; that is, the roles are adapted over time to match the capabilities of the team members. In software agent organizations, where this kind of adaptation is difficult, it must be considered as a key part of the planning process for team design.

3. Lessons from Prior Work

3.1. The Common Prototyping Environment

In a previous project, we developed the Common Prototyping Environment (CPE) for the ARPA/Rome Laboratory Planning Initiative (ARPI), a government-sponsored research initiative in planning and scheduling technologies [6]. This environment was used to tie together AI-based planning, scheduling and simulation systems to address logistics planning and scheduling problems. The CPE Testbed was designed to run experiments consisting of multiple trials that tested the speed and effectiveness of new technologies, both singly and in combinations. Trials consisted of running problems with different 'modules' specified in advance and playing particular roles. Modules for each role were selected based on their service class, where each service class defined the set of messages that all members of that class could handle. Each module, when was "pre-wired" with **meters** so that information could be collected about execution and communication behavior and timing.

Twelve different AI-based planning system modules, from ten research groups, interacted via the CPE's distributed communications layer called KNET, an early variant of KQML implemented using CRONUS, a predecessor of CORBA. Each module was classified by service class, (i.e., as a planner, temporal reasoner, scheduler, knowledge server, etc.) and communications protocols for interactions between classes of modules were established by negotiations among the implementers. Figure 2 shows the patterns of communications established between the service classes for those twelve modules. The content of all communications were represented using KRSL, an AI frame language built on ISI's LOOM but adding a number of planning ontology primitives. Each individual module was responsible for translating message content into and out of KRSL for the classes of messages it could handle.

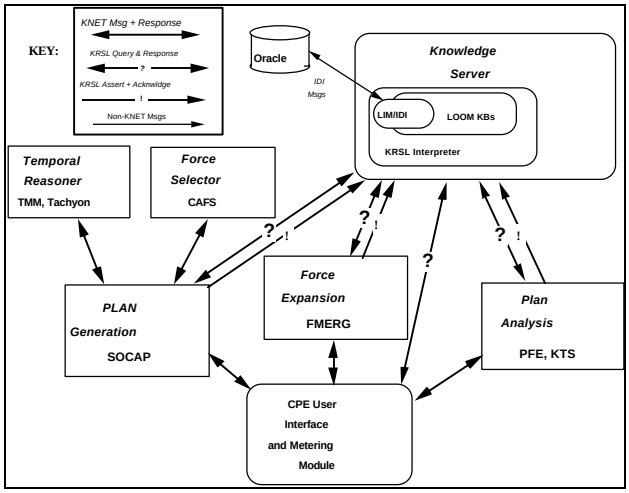

Figure 2: Communications patterns in the CPE

Although this was essentially a first attempt to get AI-based planning systems built independently to interact in a distributed environment, it is somewhat inappropriate to call their CPE incarnations "agents" for several reasons.

- Although many of the modules embodied explicit domain knowledge, none maintained a persistent, evolving, internal knowledge base of ground-level assertions, except the CPE Knowledge Servers, which maintained all plan-related information in internal LOOM KBs, or RDBMS.
- None of the modules supported inter-agent dialog (as distinguished from acontextual message-response communications),
- The modules did not actively monitor their environment, but rather waited passively for input messages.

In the CPE, most modules acted strictly as servers; not initiating messages to other agents, and responding synchronously to each message after some amount of processing. Each module that did initiate messages to other modules (primarily the planner, SOCAP) did so deterministically, sending messages to other agents at specific points in their algorithms (*e.g.*, to test the consistency of a temporal constraint network after refining a plan one full level), rather than as the result of a context dependent reasoning process.

In terms of agent attributes like those defined by Etzioni and Weld [8], these modules were not agents because they were not *reactive* (sense their environment), *autonomous* (goal directed, self-starting), *collaborative* (engage in dialog), *adaptive* (persisting, evolving internal state), or *mobile.*

On the other hand, the overall system certainly had a number of attributes that *could* also appear in agent-based systems. Each module had functional capabilities whose complex external information requirements were carefully represented in a shared domain ontology so that a consistent communications language representation could be used. Collectively, the modules solved large-scale planning problems by a combination of a number of specialized automated reasoning techniques. Certainly, many systems which employ a set of distributed software components using automated reasoning techniques and communicating symbolically have been *called* agent-based systems.

Whether or not we call the CPE an agent-based system, we feel that some practical lessons were learned from this experience that do apply to collaborative software agent team development. First, developing a single, consistent ontology (including a large number of domain terms) for communications among a large group of agents is extremely difficult, and probably unnecessary in practice. Functionally specialized agents will generally require only the capability to communicate with a restricted range of other agent types. Thus, agents who can interpret a set of partially overlapping ontologies for the purposes of communicating with agents of different types are more likely to be able to work and adapt in an open, dynamic agent environment. A key challenge for *planning* such coordination is characterizing the capabilities of individual agents not only by their individual functional capabilities, but also by the classes of other agents they can interact with.

Second, to be robust, agent-based systems must be able to handle both productive and unproductive responses from other agents. The CPE was very brittle in this sense. Failures to produce answers, or time-outs for communications reasons would essentially bring the system to a halt.

Third, although perhaps effective for knowledge consistency maintenance, holding all shared state in information server agents (*e.g.*, a knowledge server) and using essentially "stateless" functional agents may be both too slow (when the functional agents require large amounts of contextual information to function, and the cost of remote communication is high) and incompatible with any kind of collaboration beyond the client-server model. Agents collaborating by engaging in a dialog about a task must be able to build up a shared contextual model of the problem being addressed, and 'know' their relative roles in solving it. At least for the duration of a joint objective, agents may also need to monitor information streams and filter and store internally information that is relevant to their specific task. They may need to continually communicate with their collaborators to maintain a shared problem focus, share new assumptions and put aside old ones. Ultimately each agent might need a capability to shift among multiple reasoning contexts (as when searching different parts of a planning space) with different, overlapping but potentially inconsistent information states, and maintain multiple conversational threads related to those different contexts[5].

3.2. Interactions between Two Planning Tools

These last points are echoed in another recent experience linking two knowledge based planning systems together - Prodigy-Analogy [22], a generative and case-based planner and ForMAT [16], an interactive case-based force deployment planning tool. In mixed-initiative planning, automated and human planners interact to jointly construct a plan that satisfies the goals in a specific mission statement. It is well known that human planners rely strongly on past planning experience to generate new plans. In simple terms, ForMAT lets users browse or query a case-base of deployment plans (essentially lists of items to move, where and when), and 'cut and paste' together new ones to address a new objective. ForMAT and Prodigy/Analogy were linked to form a mixed-initiative system where Prodigy acts as a planning support agent, providing advice on this process. [23].

Events (ForMAT user input)	**Messages to Prodigy**	**Prodigy Responses**
Intel, Objectives, Resources given	New Objective, Assumptions, Resources	Information stored.
User selects to deploy F16 squadron	New Subgoal: DEPLOY F16	Subgoal incorporated. Suggests INCLUDE F16 support unit.
User copies fighter support unit into plan	New Subgoal: DEPLOY F16 Support Units	Subgoal precondition check fails. Suggests REJECT task.
New Intel arrives	New Info.	Replan. Propose revised forces

Table 1: Interaction between planning agents

In the combined system, neither ForMAT nor Prodigy/Analogy possesses full knowledge of the other system's state, but messages between the two maintain a shared partial state. A set of pre-established message patterns is used to keep the other current, and suggest actions (*e.g.*, to modify the plan). Table 1 describes some

of these messages and their effects. In this figure, ForMAT interacts with the user. Some user events, such as the user specifying new goals (objectives) are relayed to Prodigy/Analogy, which responds by suggesting possible actions. For example, the goal to deploy an F-16 aircraft produces a suggestion to bring along a support unit. The selection of a support unit that is not available produces a suggestion to reject that part of the plan. When new information arrives that changes planning assumptions, this information, when relayed to Prodigy, causes it to replan using the previous plan as a guide.

This style of interaction among agents and users is what we are seeking to extend and generalize in our current effort. The interactions between ForMAT and Prodigy/Analogy are a step in the direction of multi-agent mixed-initiative planning, where an interface agent (in this case, ForMAT) plays the role of user-directed plan editor, supported by an agent doing automated planning support (Prodigy/Analogy). Our new system, TEMPLATES (Team Execution Manager and Planner for Agents by Task Editing System), will carry this further, by providing a suite of plan editing and visualization tools that supports both domain planning and explicit tasking of other agents to support that planning.

Taken together, the examples of the CPE and ForMAT/Prodigy raise some of the issues that we face in coordinating intelligent agents. The Common Prototyping Environment allowed users to select which "agents" would participate in the current assigned task, and monitor the results. It also taught us how difficult it is to populate such an environment with distributed agents or modules that can communicate effectively about complex representations like plans in order to solve problems. Each pair of planning modules that was capable of interacting needed to communicate in a shared representation language, different than their internal one, and needed to access contextual information and intermediate planning products. Each agent sending a message needed to know what kinds of information the receiver needed to be able to respond successfully, and the appropriate syntax to communicate that information.

Although ForMAT and Prodigy do maintain state and communicate in an ongoing fashion, many of the same issues were considerations in designing their interactions. None of these systems have general-purpose reasoning capabilities (planning systems may plan, but they don't do much general inference), and each has limited abilities to represent and reason with information. As a result, their information exchanges with other agents are restricted largely to the intersection of their (explicit or implicit) internal ontologies. Going outside that intersection leads to misinterpretation.

We believe that these issues will not go away as we move to more dynamic agent team collaborations. We will need to make some strong assumptions about the nature of inter-agent communications for any group of agents that actively collaborates; assumptions that are based on the ontologies handled by each agent, and their inferential capabilities. As most agents will not be able to reason about and compare their own ontologies or domain concepts to those of other agents they will be interacting with, the job of circumscribing a shared subset of the ontologies of two agents for the purposes of defining their inter-agent communications must be done in advance. We see this as the role of an administrator defining a pool of agents from which task-specific teams can be drawn. Walker and Woolridge [24] termed this *off-line design* of agent conversation policies.

By assuming a mixture of approaches including shared representation languages for clusters of agents designed to work together frequently, and translators, perhaps residing in mediation agents, for communications between agents that work together less frequently (or normally work in different agent pools), we believe that we can minimize the explicit coordination of communications protocols during dynamic team planning. This is necessary if users are to manage teams from a strictly domain-oriented standpoint, without needing to understand the low-level communications language issues as well.

Unfortunately, during collaborative agent team task execution, users may well have to deal with inter-agent communications problems. For example, the dynamic introduction of information mediation agents implies potential new sources of communications failure, delay and miscommunication. If the team management environment is to help users to understand and deal with communications breakdowns, good monitoring and communications tracing must be combined with effective visualizations so that users can trace causes of failure, and correct the problems or suggest alternative taskings.

Since the execution of tasks is extended in time, and deadlines for major objectives can span both agents and tasks, monitors must be able to manage time allotments for tasks performed by individual agents. All agents should be capable of responding to periodic requests for information on their processing status and expected completion time. Agents must also be able to report problems and failures gracefully, and give reasons if possible. Agents reporting problems should at least distinguish between communications failures, information interpretation problems, lack of information, inability to complete a task, and partial solutions.

To enable agents to routinely report on their status during execution, each agent should have procedures for calculating and reporting on the expected time to complete the tasks it knows how to perform. If the tasks performed

by an agent are broken down into a sequence of subtasks, status reports may indicate progress through these phases.

3.3. Discussion

The current generation of software are predominantly either (1) designed to interact directly with human users and produce results that are only interpretable by humans (*e.g.*, information gathering web-crawlers that return uninterpreted information) or (2) essentially distributed object systems with agent communication wrappers that precisely define their protocols for use in an agent environment. Many of the idealized attributes found in anthropomorphic characterizations of software agents; things like autonomy, goal directedness, ability to communicate (and interpret communications), exhibit initiative, reason, plan, adapt and learn, are ascribable at best only in very limited forms to most current-day agents. The next generation of software agents must extend the capabilities of both of these classes of agents to include more robust, extended communications with, at least, known classes of other software agents, if they are to be capable of acting as members of agent teams organized dynamically.

As we have tried to argue, successful planning for the activities of teams of agents requires good characterizations of all available agent capabilities in both functional and potentially domain-specific terms. The challenge is to enable users to make effective use of the capabilities that are provided by the available agents, and to help manage the execution process of those agents so that the user's objectives are achieved.

For our work, we are assuming that agent team planning takes place with pre-specified agents having appropriate kinds of capabilities for inclusion on teams within the domain of concern. Each agent will have known capabilities to interact with other specified agents in the same pool. By focusing on the task of planning the composition and tasking of teams of agents chosen from such collection of agents, we can address issues involved in agent team configuration and management.

4. Detailing the Capabilities of Team Players

For coordination management of teams, representations of agent capabilities and limitations must be made explicit. They should be user interpretable, so that both human users and their personal planning/management support agents can reason interactively about agent capabilities to handle specific domain tasks. As tasks are identified and delegated, capabilities models are used to anticipate performance, and to reason about failures of individuals and the team as a whole.

Each class of agents in an "agent pool" will be characterized by a functional profile consisting of information about the capabilities of the agent, its pre-existing internal knowledge and communications abilities. For planning purposes, most of the profile can be organized around characterizations of the tasks the agent can perform, and the goals that it may pursue. Profiles must include information about the other agent classes it can communicate with, the tasks it can request of those other agents, and the circumstances (which of its own tasks) might require those interactions. This information is needed to properly plan tasks for agents or assign them to team roles, and to identify when tasked agents may optionally need to request information from remote agents, perhaps through mediators.

For each class of agents supporting a task, there needs to be a task description that includes that task's information requirements and products. This must be accompanied by information about the representation language(s) used to communicate with the agent, and the domain ontology used by the agent to reason about the task. Agent task characterizations should, where possible, include decompositions into subtasks with explicit temporal/functional dependencies, and explicitly include expected patterns of communication with other agents. Characterizations of expected communications should delineate informational requests, task delegations to other agents, and responses to status requests from task monitoring agents.

While we speak of communications in terms of simple, single messages and responses, most inter-agent communications will, in fact be extended sequences of messages requiring some reasoning by the agent before each response. When we represent that an agent may initiate a communication using some performative, we are implying that the recipient agents might respond multiple times, up to and including starting an extended negotiation. Even the simplest kinds of exchanges will generally involve message sequences. A *request* to do a task might be *accept*ed or *reject*ed, and if accepted might be followed by multiple *status* messages, additional information requests, delegations of subtasks to other agents, and finally a message that the task has been *completed* or *failed*. Similarly, a *request* for information might be *accept*ed as a task or not, with the final result being an *inform* or *dont-know* (fail), again with the possibility of intermediate status reports, and requests to other agents. Part of our effort will be the development of procedural task models of the various stages of communication and internal processing that each agent goes through in extended communications with other

agents to perform various tasks. These are precisely the kinds of models that OMAR can directly simulate.

Table 1 sketches a partial taxonomy of simple performatives and their potential responses based on the type of message content (tasks vs. informational). It shows some basic dialog patterns that will appear repeatedly in more domain specific task and communications models.

Performative	Response to Info	Response to Task
request (Task or Info)	accept | reject | inform | status	accept | reject | counterpropose | recommend | status
perform (a command)		same as above, but no **reject ...**
inform	accept | reject	
assume	accept | reject	
suggest (plan elt or modification)	accept | reject (if authorized) concur | disagree	

Table 2: Responses to performatives

5. Managing Dynamic Agent Teams

Effective utilization of *agent teams* requires planning, not just to decide which agent should do which task, but to lay out the tasks necessary to achieve the domain-specific objectives effectively, and in a timely fashion.

In reality, this planning is interleaved with execution in an ongoing process. Tasks may not be able to be assigned until preliminary information gathering has occurred, and that information may have to be reformulated to be used by the agents requiring it. Thus, agent tasking plans must be living documents, changing from moment to moment as tasks are accomplished or the situation changes.

In large planning organizations, such as the military logistics and transportation commands, large numbers of people manage to continuously plan with information available to them through, at best, shared databases, and often with just telephonic or email communications. Problems are often due to the effort required to find and coordinate key pieces of information in the multiple incompatible databases and the great volume of textual messages that must be handled. Different “cells” in the organization do not always see the impact of the information they possess on other aspects of the organization. For this to change so that fewer breakdowns in communication and coordination occur, software agents must become part of the organization [19].

As with the people in the organization, there may be pre-defined roles for software agents, and default communications channels to gather and disseminate information to others in the organization. Groups of functionally similar agents may manage different pieces of domain information (in military planning organizations, responsibilities divide along the lines of the kinds of resources of to be managed, by classes of tasks, such as air cargo transport vs. air refueling).

In this context, where agents have *organizational* roles, and well represented capabilities, we believe that mixed-initiative task planning may be accomplished using a combination of interactive hierarchical task network (HTN) and case-based planning [10, 13, 22] techniques. Constraint-based scheduling techniques may be used to maintain dynamic task plans that manage the available agent resources, and generate expectations for intermediate task completion.

Team planning support agents helping a team manager to in an ongoing planning and execution environment must help coordinate the team’s efforts during execution. If the human manager is continuously elaborating and revising the team’s objectives, the planning support agent must be able to identify when tasks or agents are no longer appropriate and must be replaced, or new taskings are required. Examples of this kind of dynamic tasking include the generation of information seeking/gathering tasks that can be anticipated while team task planning is still ongoing, the selection of alternative agents to perform tasks when the agent originally tasked is busy or cannot support the task for other reasons, and the generation of new tasks when an agent can only provide partial results. Although some of these issues have been addressed specifically for information agent task planning by the SIMS project at ISI [3, 12], much remains to be done.

6. Simulating Agent Team Behavior

Our exploration of approaches to mixed-initiative team management requires the existence of a variety of kinds of agents, with some capabilities (such as status reporting, knowledge reformulation) that don’t generally exist yet. To explore effective styles of interaction between humans and the planning and execution management support agents that we are building, we are developing an environment where we can simulate a variety of software agents executing the multi-agent task plans our users create, so that we can evaluate interaction effects. We see this as a way to explore effective approaches to human team leaders collaborating with and directing agent teams.

The OMAR (Operator Model ARchitecture) agent simulation environment [7, 9] is well suited to supporting our proposed efforts. Developed at BBN over the last ten years, OMAR is an on-going research program developing simulation tools and a support environment for studying and evaluating human performance models, workplace/workflow models and operating procedures for

new equipment being considered for use in complex operating environments. The current implementation of OMAR's simulation environment traces its roots to ACTORS [4, 11], an early MIT research effort in building a distributed computing environment. A predecessor of OMAR was developed for the original Semi-Autonomous Forces component of SIMNET [1, 2], and was used to simulate the coordinated actions of teams of agents (tanks and their commanders) on a simulated battlefield.

The current OMAR environment provides an extensive set of tools to support the development and debugging of TEMPLATES agents and agent organizations, as well as data analysis display tools for the evaluation of TEMPLATES simulation runs. Distributed OMAR, the version of OMAR currently under active development, operates with simulated (and real) agents at multiple sites in a distributed-object computing environment.

The human capabilities and attributes that we emulate in OMAR human performance models are very much like those that we would like to have in software agents. OMAR agents are capable of goal-directed proactive behaviors, defined by subtask networks. At the same time, they can respond appropriately to sequences of events evolving in complex situations. OMAR agents communicate with other agents and the human operators of the system. Agent goals and procedures are defined to anticipate failures and include alternate paths to success.

OMAR agents anticipate and respond to events and communications using *signals*. Signals are predicate assertions that activate the triggers in agents waiting on signals with a particular pattern. An agent's sensitivity to a given signal is established by the agent's procedures reaching a point where it waits for that pattern of signal.

OMAR agents typically pursue several active goals concurrently. Some of an agent's goals can be independent of one another, while others may have explicit dependencies. Plans may be required to execute sequentially or they may be allowed to execute in parallel. Complex execution dependencies are defined using signals. In the OMAR modeling environment, the simulator emulates parallel procedure execution. In real world applications, the OMAR simulator executive manages OMAR agents and real remote agents pursuing multiple goals in parallel, dynamically computes task priorities, and mediate conflicts between contending tasks.

7. Agent Team Evaluation

The evaluation of alternative software agent organizations and of mechanisms for software agent command and control by human users is an important thrust of our work**.** The TEMPLATES environment is being designed such that we can experiment with the kinds of capabilities that we think users will require to control their software agents, while enabling us to perform experiments on the effectiveness of alternative software agent organizations in various task contexts and given different levels of agent capability. These experiments will address such questions as "What are effective agent team organizations for different classes of activity?" "What kinds of agent monitoring are most effective?" "What kinds of agent and management capabilities are needed to support retasking?" We plan to do a series of experiments aimed at uncovering such things as the benefits and problems of different styles of communications between agents (*e.g.* directed, mediated communications vs. shared ontology/knowledge sources), hierarchical or matrix organizations of agent teams, larger scale market-driven agent organizations, and the utility of "multi-talented" agents vs. many single-function agents.

8. Summary

Our work is directed at developing an effective near-term relationship with agents, by prototyping a human user-centric view of how agents will be used. We anticipate that, for some years, software agents will most often act 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.

We are developing the TEMPLATES system to support people in the role of agent team manager, as they are tasking and monitoring agents. The TEMPLATES' plan editing environment will provide a set of plan editing tools that can be composed or combined with domain-specific interfaces to provide human users with mechanisms to describe a kind of "strategy to task" decomposition of their objectives in domain-oriented terms.

TEMPLATES' planning support agents will continually assist human managers in refining objectives into tasks, finding appropriate agents for tasks, and ensuring that support is in place for agents to communicate with each other and the user. They will task information gathering and information reformulation agents as needed to provide tasked agents with the data that they need in the proper form. They will also ensure that team plans include monitoring agents to keep track of the progress of agents in their assigned tasks, and report back when issues and problems arise.

By building TEMPLATES on the OMAR simulation environment, we can experiment with and evaluate alternative models of interactive team management and explore appropriate team organizations for different tasks.

We believe that the TEMPLATES will help human users acclimate to the usage of agent-based systems through tools that it will provide human team leaders for understanding, tasking, and controlling pools of agents. For the military user, the TEMPLATES style of interface should prove valuable in supporting a variety of problem solving areas in intelligence, logistics and command and control situations, as well as for training.

9. References

[1] Abrett, G and Burstein, M. and Deutsch, S., "An Environment for Building Goal-directed, Knowledge-based Simulations", BBN Technical Report 7062, 1989.

[2] Abrett, G., "Planning by Autonomous Agents with Many Concurrent Goals in an Elaborate Simulated World"' Proceedings of the IEEE Conference on Planning in High Autonomy Systems, Cocoa Beach, FL, 1991.

[3] Arens, Yigal, Chee, Chin Y., Hsu, Chun-Nan and Knoblock, Craig A., "Retrieving and integrating data from multiple information sources", International Journal on Intelligent and Cooperateie Information Systems, 2(2):127-158, 1993.

[4] Agha, G. A.,. Actors: A model of concurrent computation in distributed systems. MIT Press, Cambridge, MA, 1986.

[5] Burstein, M. and McDermott, D., "Issues in the Development of Human-Computer Mixed-Initiative Planning",. In B. Gorayska and J.L. Mey (eds.), Cognitive Technology, Elsevier, 1996, pp. 285-303.

[6] Burstein, M. Schantz R., Bienkowski, M.A., desJardins, M.E., and Smith, S., "The Common Prototyping Environment: A Framework for Software Technology Integration, Evaluation, and Transition," Advanced Planning Technology, A.Tate (ed.), The AAAI Press, Menlo Park, CA, May 1996, ISBN 0-929280-98-9. (Also in IEEE Expert, February, 1995.)

[7] Deutsch, S. E., & Adams, M. J., The operator-model architecture and its psychological framework. 6th IFAC Symposium on Man-Machine Systems. MIT, Cambridge, MA, 1993.

[8] Etzioni, O. & Weld, D. S., Intelligent Agents on the Internet: Fact, Fiction, and Forecast. IEEE Expert 10(4): 44-49, 1995.

[9] Freeman, B. (1997). OMAR User/Programmer Manual, Version 2.0. BBN Report No. 8181. Cambridge, MA: BBN Corporation.

[10] Hammond, K., "Chef. A Model of case-based-planning", In The Proceedings of the Fifth National Conference on Artificial Intelligence, AAAI Press, August 1986.

[11] Hewitt, C. & Inman, J., "DAI betwixt and between: From 'intelligent agents' to open systems science", IEEE Transactions on Systems, Man, and Cybernetics, 21, 1409-1419, 1991.

[12] Knoblock, C.A. and Ambite, J. L, Agents for information gathering. In (Bradshaw, Ed.) Software Agents. MIT Press, Cambridge MA, 1991.

[13] Leake, David B., Case-Based Reasoning: Experiences, Lessons and Future Directions, MIT Press, Cambridge MA, 1996.

[14] Malone, T.W. "Modeling Coordination in Organizations and Markets", Management Science, 33(10), 1986.

[15] Malone, T.W., and Crowston, K., "Toward an interdisciplinary theory of coordination", Center for Coordination Science Technical Report No. 120, MIT, Cambridge, MA, 1991.

[16] Mulvehill, Alice M., "Reusing Force Deployment Plans", AAAI Fall Symposium on Adaptation of Knowledge for Reuse, MIT, Cambridge Mass, November 10 - 12, 1995.

[17] Norman, Donald. A., "How people might interact with agents", In Bradshaw (Ed.), Software Agents. MIT Press, Cambridge MA, 1997.

[18] Orasanu J. & Salas, E., "Team decision making in complex environments",. In Klein, G. A., Orasanu, J., Calderwood, R. and Zsambok, C. E. (Eds.) Decision Making in Action: Models and Methods. Ablex, Norwood, NJ: 1993.

[19] Pan, J. Y. C. & Tenenbaum, J. M., "An intelligent agent framework for enterprise integration", IEEE Transactions on Systems, Man, and Cybernetics, 21, 1391-1408, 1991.

[20] Salas, E., Dickinson, T. L., Converse, S. A., & Tannenbaum, S. I., "Toward an understanding of team performance and training", In Swezey, R. W. and Salas, E. (Eds.) Teams: Their Training and Performance. Norwood, NJ: Ablex Publishing Corporation, 1992.

[21] Shneiderman, B., "Direct manipulation vs. agents: Paths to predictable, controllable, comprehensible interfaces", In Bradshaw (Ed.), Software Agents. MIT Press, Cambridge MA, 1997.

[22] Veloso, M., Learning by Analogical Reasoning and General Problem Solving. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1992. Technical Report CMU-CS-92-174.

[23] Veloso, Manuela M., Mulvehill, Alice M. and Cox, Michael "Rationale-Supported Mixed-Initiative Case-Based Planning", Proceedings of AAAI , 1997 .

[24] Walker A. and Wooldridge, M., "Understanding the emergence of conventions in Multi-agent systems", In V. Lesser (Ed.) Proceedings of the First International Conference on Multi-Agent Systems, AAAI, Menlo Park, CA, 1995.

[25] Wooldridge, M. & Jennings, N. Agent theories, architectures, and languages: A survey. In M. Wooldridge & N. Jennings (Eds.), Intelligent agents, Springer-Verlag, Berlin, 1995.