Executing Decision-theoretic Plans in Multi-agent Environments (Extended Abstract)

Mike Williamson and Keith Decker and Katia Sycara
The Robotics Institute, Carnegie-Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
(mikew,decker,sycara)@cs.cmu.edu

April 15, 1996

1 Introduction

An agent that carries out plans in a realistic domain faces many difficulties. Particular difficulties arise, though, when an agent participating in a *cooperative, multi-agent* environment is executing *decision-theoretic* plans. This paper describes these specific difficulties, and discusses potential avenues to their solution.

By a "cooperative, multi-agent environment," we mean a system in which agents interact to collectively solve problems. In such an environment, many of the "actions" that an agent can perform are in fact speech acts, which induce some other agent to act on the first agent's behalf. In some cases, the use of speech acts can simplify an agent's planning process by off-loading some of the planning effort. For example, if you want an airplane ticket it's much easier to call a travel agent than personally to go through the process of calling individual airlines, finding out schedules, comparing prices, making reservations, etc. But speech acts can also be problematic, because they magnify the existing problems faced by a single agent reasoning about its own actions:

Uncertainty An agent must rely on models of its actions when formulating plans, and these models may be inaccurate or incomplete. In the case of speech acts, the underlying models are of *another* agent's abilities, so the models are likely to be even less accurate and less complete. Indeed, that's part of the reason for engaging other agents: they possess knowledge or abilities that you don't. The result, though, is that one cannot be sure of the eventual consequences of a speech act.

Dynamism A central problem faced by many agents who plan is that the world is constantly changing. This is especially true in the case of multi-agent environments, since agents are inherently dynamic entities. They appear and disappear, and their abilities may change over time. When you call your travel agent to arrange a trip, you might find that he has gone out of business. Or much worse, your travel agent might go out of business *after* you request him to arrange your trip, without actually making any arrangements. This raises a significant plan monitoring difficulty. Since the effects of a speech act can be spatially and temporally distant from the act itself, how can an agent tell if a plan is proceeding as intended?

We construe "decision-theoretic plan" quite broadly to mean a plan intended to be "good" with respect to some explicit measure of quality (i.e. a value or utility function).[1] The notion of goodness could be interpreted strictly as the optimization (over all possible plans) of the utility function, or more loosely in terms of approximate or near optimization. In any case, decision-theoretic plans are distinguished from "classical" plans, which have only a meager, implicit conception of quality: that a plan is "good" if and only if it achieves a specific goal, and "bad" otherwise." Consequently, the decision-theoretic planning process is inherently relative; whether or not to use a particular plan depends not just on the utility of that plan, but on the utilities of alternative plans as well. This relativity complicates the process of executing and monitoring decision-theoretic plans. As in the classical case, an agent must ensure that its ongoing plans are having their intended effect. But a decision-theoretic agent must also watch for changes that would make alternative plans better than the current one. For example (assuming that money has utility), a plan to use a travel agent might seem less appealing if you find out about a ticket consolidator offering deep discounts that aren't available through your travel agency. From a classical standpoint the plan to use the travel agent is no worse than before, but from the decision-theoretic perspective it is no longer the best.

When agents execute decision-theoretic plans in multi-agent environments, the problems mentioned above are compounded. In the remainder of this paper, we will give concrete examples of the problems that arise, and describe various approaches to their solution. Our examples are drawn from our work on WARREN, an Internet-based, cooperative, multi-agent system for information gathering and decision support, applied to the domain of financial portfolio management. We will begin with a brief description of WARREN.

2 WARREN: Multi-agent Portfolio Management

Although we are developing domain independent agent architectures and organizational structuring techniques, we are evaluating this work in the context of a complex, real environment. For our example task environment, we have chosen the task of providing an integrated financial picture for managing an investment portfolio over time, using the information resources already available over the Internet. This task environment has many interesting features, including:

- the enormous amount of continually changing—and generally unorganized—information available;

[1] To be more precise we should call such plans "decision-theoretically generated plans," since it is the process of their creation—rather than the plans themselves—that is decision-theoretic, but we hope that our meaning is clear.

- the variety of kinds of information that can and should be brought to bear on the task (market data, financial report data, technical models, analysts' reports, breaking news, etc.);

- the many sources of uncertainty and dynamic change in the environment;

- information timeliness and criticality features that present the agents with soft real-time deadlines for certain tasks; and

- resource and cost constraints—not all data are available for free, or are of equivalent quality.

In the portfolio management domain, people today solve this problem either by reviewing information (collected and integrated by other people) at the end of long periods, or by tracking just a few narrow information sources. Either choice can limit return on investment and/or increase the amount of risk to which the portfolio is exposed. Alternately, people pay others to coordinate a team that tracks and integrates the appropriate information. This motivates our use of a multi-agent organization where a collection of intelligent software agents inter-operate to collect, filter, and fuse information from distributed, network-based information sources.

The WARREN system consists of a collection of agents that cooperate to perform portfolio management services for one or more users. WARREN agents work on multiple tasks in the service of multiple concurrent objectives. At the highest level these tasks include the (continuous) portfolio management objectives of user profiling, asset allocation, risk management, and portfolio monitoring. At the level of monitoring individual portfolio assets, tasks include monitoring an asset currently being held (should we continue to hold it? sell some or all of it?), or monitoring the purchase or sale of an asset. At the lowest level are tasks associated with information gathering, which involve direct interaction with Internet-based resources such as web pages or news servers.

2.1 Organizational Structure

Our system comprises three basic kinds of agents: *interface agents* interacting with each individual user, *task agents* involved in the processes associated with arbitrary problem-solving tasks, and *information agents* that are closely tied to a source or sources of data (see Figure 1). Each user would have his or her own interface agent, although many task and information agents may be shared. Each agent in WARREN continually interleaves planning, scheduling, coordination, and the execution of domain-level problem-solving actions. (We will describe the internal architecture of our agents in Section 2.2.)

The organizational structure of WARREN arises dynamically as agents request and supply services to one another.[2] Agents do not initially know about other agents capabilities (or physical locations) and must supply and obtain this information using abstract specifications ("advertisements" and advertisement queries). This information exchange is managed by a well-known "matchmaker" agent. The system's organization can change over time in response to the arrival of new agents, the departure of agents, and the creation of new

[2] WARREN agents use KQML for inter-agent communication [5].

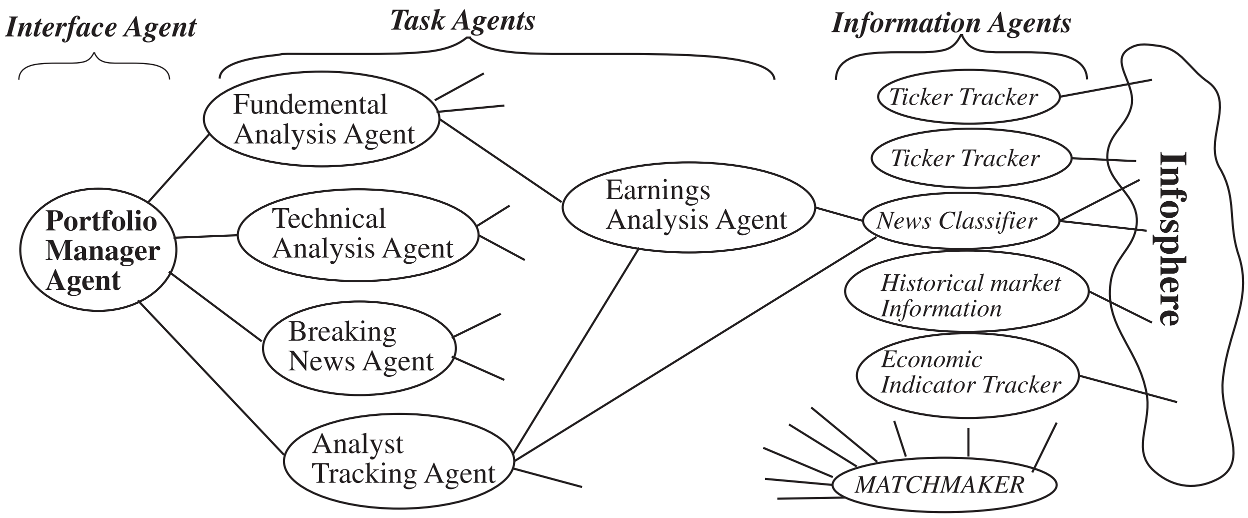

Figure 1: Some of the agents involved in the WARREN financial portfolio management system.

relationships between agents (as the user and agents respond to new information or changes in the environment). However, the organization can also remain relatively static for extended periods (for example, while monitoring currently held investments during stable market periods). Such multi-agent systems can compartmentalize specialized task knowledge, can re-organize themselves to avoid processing bottlenecks, and can be built expressly to deal with dynamic changes in the agent and information-source landscape.

2.2 Internal Agent Architecture: Planning, Scheduling, and Action Execution

An agent's possible behaviors are determined by its internal agent architecture, i.e. the set of generic software components for knowledge representation, agent control, and interaction with other agents. The generic software components are common to all agents, from the simple information agents to more complex task agents and interface agents. Our current agent architecture is an instantiation of the DECAF (Distributed, Environment-Centered Agent Framework) architecture [2] (Figure 2).

The control process for agents includes steps for planning to achieve local or non-local objectives, scheduling the actions within these plans, and actually carrying out these actions. These control processes can be thought of as occurring concurrently, although they are serialized in our implementation. The agent executes an initialization process upon startup that bootstraps the agent by giving it initial objectives to poll for messages from other agents and to advertise its capabilities.

New objectives are created by the arrival of service requests from other agents, or in response to internal events (such as the need to gather information). The objectives supplement traditional symbolic goals with additional information regarding the relative importance of goals, the value of partially satisfying goals, and the agent's preferences about other (non-goal) aspects of the world. The objectives collectively provide a basis for the definition of

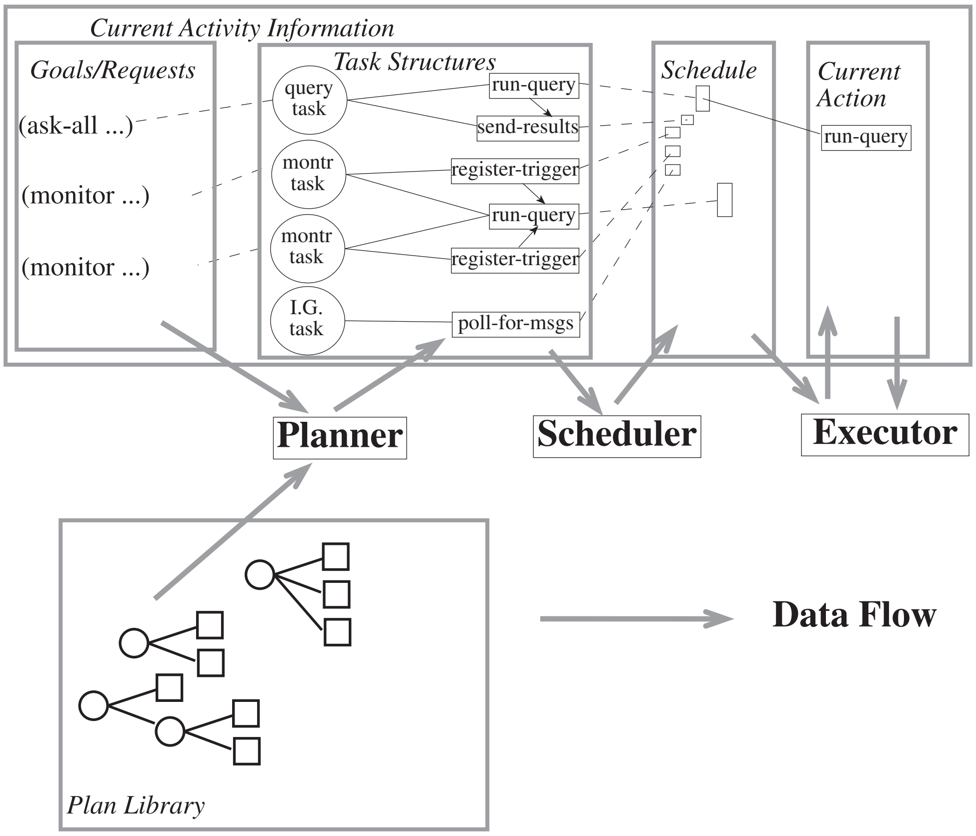

Figure 2: Overall view of an agent's internal architecture.

the agent's utility function.

The agent's hierarchical task network (HTN) planning process [3] takes as input the agent's current set of objectives, the current set of task structures, and a library of task reduction schemas. The task reduction library encodes both agent-specific knowledge (such as how to manage a portfolio, or how to access a particular information source), and shared, social knowledge (such as how to interact with other agents). The planning process modifies the current set of task structures—by removing tasks, further reducing existing tasks, or instantiating new tasks—to account for any changes to the agent's objectives. Task reduction is incremental, and can be interleaved with execution.

The agent's scheduling process takes as input the current set of instantiated task structures and decides which basic action, if any, is to be executed next. This action is then identified as a fixed intention until it is carried out by the execution process. Execution of an action can have internal effects on the agent's state (such as the creation of new objectives), and/or external effects on the environment or organization (such as writing a file or sending a message to another agent).

3 Plan Execution Issues in Warren

In this section, we will describe several sources of difficulty that arise when a decision-theoretic agent executes plans in a cooperative, multi-agent environment, providing concrete examples from the Warren system.

Agents in Warren often rely on other agents to provide services that assist them in performing their own duties. For example, in order to carry out its primary task, a portfolio management agent needs to know the current prices of the securities held in the portfolio. It could plan to obtain this stock price information by enlisting the services of a stock ticker agent. The portfolio agent's plan would include sending a request to some stock ticker agent to monitor a security. Thereafter, the ticker agent would periodically send back updated prices and other financial information.

A central planning decision faced by our agents is the choice of *which* other agents to enlist. Often there will be many alternative candidates that—although they provide the same basic services—differ in cost, reliability, responsiveness, and other measures of service quality. We use the term *capabilities* to refer to a comprehensive description of: 1) the services provided by an agent, 2) any restrictions or limitations on those services, and 3) appropriate (service-specific) measures of the cost or quality of the service. For example, Warren currently has several different stock ticker agents, each drawing its information from a different Internet source. The capabilities of ticker agents can be characterized along a number of dimensions:

timeliness Stock quotes are extremely time-sensitive data, whose value quickly decays. Many web sites offer 15 or 20 minute delayed stock quotes at no cost. Other sites offer "real-time" quotes on a subscription basis.

cost The monetary costs associated with obtaining stock quotes vary from free to quite expensive. In some cases, real-time quotes are bundled with packages including other financial services.

accuracy At least one site has been observed to return incorrect stock quotes.

responsiveness Some of the web sites offering stock quotes are very popular, and response times degrade seriously during times of peak network usage.

completeness Most quote services provide other useful financial information, such as price/earnings ratios or ex-dividend date. The actual information provided varies widely from site to site. Some sites provide graphs showing the price of the stock over long periods of time.

scope Some services provide quotes only on equity securities traded on the major exchanges (NYSE, AMEX, NASDAQ). Other services might cover regional exchanges (e.g. Los Angeles, Vancouver), foreign exchanges (Hong Kong, London), or other kinds of securities (futures, options, mutual funds).

reliability Like all other web sites, the stock quote sites are subject to occasional outages, thus rendering the associated stock ticker agent temporarily unable to provide its service. Moreover the agents themselves might suddenly disappear because of system crashes, network failures, or bugs in their code.

A portfolio agent, then, is faced with deciding between various ticker agents based on their capabilities and its own preferences. If we assume (for a moment) that the portfolio agent has complete and correct knowledge of the capabilities of all ticker agents (and an adequate model of its own preferences[3]), then the choice of ticker agent is a straightforward decision-theoretic planning problem. But it is not *just* a planning problem, though; there are several characteristics of multi-agent environments that require the choice of partner to be continuously monitored during plan execution.

Changing abilities of agents A principle characteristic of intelligent agents is that their capabilities are prone to change over time. To continue the example of stock ticker agents, the underlying web site might begin carrying quotes for a new exchange, thus expanding the services that the agent could provide. Or response time could degrade because of increased loading on the quote server, the network, or the ticker agent itself. The portfolio agent must somehow monitor the capabilities of the ticker agent that it selected to be sure that it is performing as expected.

A deeper problem is that the portfolio agent also needs to monitor for changes in the capabilities of agents that it did *not* select. Imagine that the portfolio agent selected ticker agent B instead of ticker agent A because A did not provide quotes on options. If agent A does begin to provide option quotes, then the portfolio agent needs to re-evaluate its decision, and perhaps terminate its relationship with B and use A 's services instead.[4]

This problem might arise even if the capabilities of the ticker agents don't actually change. If the portfolio agent had based its decision on an *incorrect* model of one of the ticker agents, and subsequently obtained an improved model, it would then need to re-evaluate its decision.

Agent appearance Closely related to the issue of agents changing their abilities is the appearance of new agents. If a new agent can provide services that are substantially better in some regard than existing agents, then any ongoing plans that involve use of the existing agents might need to be revised. For example, imagine that a portfolio agent had decided to use a particular ticker agent that provides 15 minute delayed quotes for free. If a new ticker agent appears which provides 10 minute delayed quotes for free,[5] then the portfolio agent may wish to switch to the new ticker agent.

Agent failure and disappearance The failure or disappearance of agents raises a different issue, more akin to the traditional problem of monitoring plans for failure. But the

[3]Eventually, the preferences of the portfolio agent must derive from those of the user: how much money he is willing to expend, and how good of a picture of his portfolio he desires.

[4]Note, however, that the decision to switch from B to A is not as simple as deciding if A is now better than B, because there might be disadvantages associated with the process of switching. The portfolio agent needs to evaluate whether the expected long-term benefits of using A instead of B would outweigh the cost of making the switch.

[5]Or even for a small cost.

traditional problems are compounded because of the distribution of responsibility to individual, autonomous agents. Consider the following scenario. A portfolio agent sends a request to a ticker agent: "Tell me when the price of IBM reaches 130." A week goes by, with no response. Has the ticker agent failed, or has the stated condition simply not occurred? Detecting the failure of such services is extremely difficult. Unfortunately, this problem is the flip side of the most significant benefit of—and major motivation for—using a multi-agent system.

4 Possible Approaches to Execution Monitoring

In the previous section, we described some issues faced by a decision-theoretic agent executing plans in a cooperative, multi-agent environment. In short, the problem is this:

> If an agent makes a plan that involves the performance of services by other agents, how can it be sure that the plan remains good in the face of changing abilities, appearance, and disappearance of those other agents?

Generally speaking, the solution is to monitor execution of the plan. We have explored several different approaches to execution monitoring, which can be seen as falling into three broad categories. While these categories are neither exhaustive nor mutually exclusive, we believe they provide a good framework for comparing alternative solutions.

4.1 Monitoring in the Plan

The first approach is to build execution monitoring activities directly into the task structures that constitute the agent's plans. Warren agents use a hierarchical task network formalism that allows the representation of plans with information gathering actions, conditional branches, and loops. This formalism also supports a unified representation of control and information flow within plans.[6] Using this representation, it is possible to define task structures that include explicit behaviors to compensate for the dynamism in the world.

For example, a common task our agents must perform is to find answers to various queries. If an agent does not itself have the knowledge to answer a query, it must turn to another agent for help. Figure 3 shows part of a generic query-answering task structure. The boxes in the figure represent tasks; "Send Query" and "Process Reply" are primitive actions which can be executed directly, while the remainder are abstract tasks. The dashed lines represent task/sub-task relationships. On the left side of each task are its *provisions*, representing information that must be supplied before the task can be executed.[7] On the right side are the possible *outcomes* of the task. Solid lines represent information flow between tasks.

Answering a query is reduced to two subtasks, finding an agent to ask and querying that agent. (Finding an agent is itself a complicated process, that involves sending a (different) query to the matchmaker agent.) "Query Agent" is reduced to two primitive actions, which

[6]Our task network formalism also incorporates features of reactive task architectures such as TCA [9] and RAPS [6]. See [10] for a complete description of our representation.

[7]Provisions are a generalization of parameters and "runtime variables" [1, 4, 8].

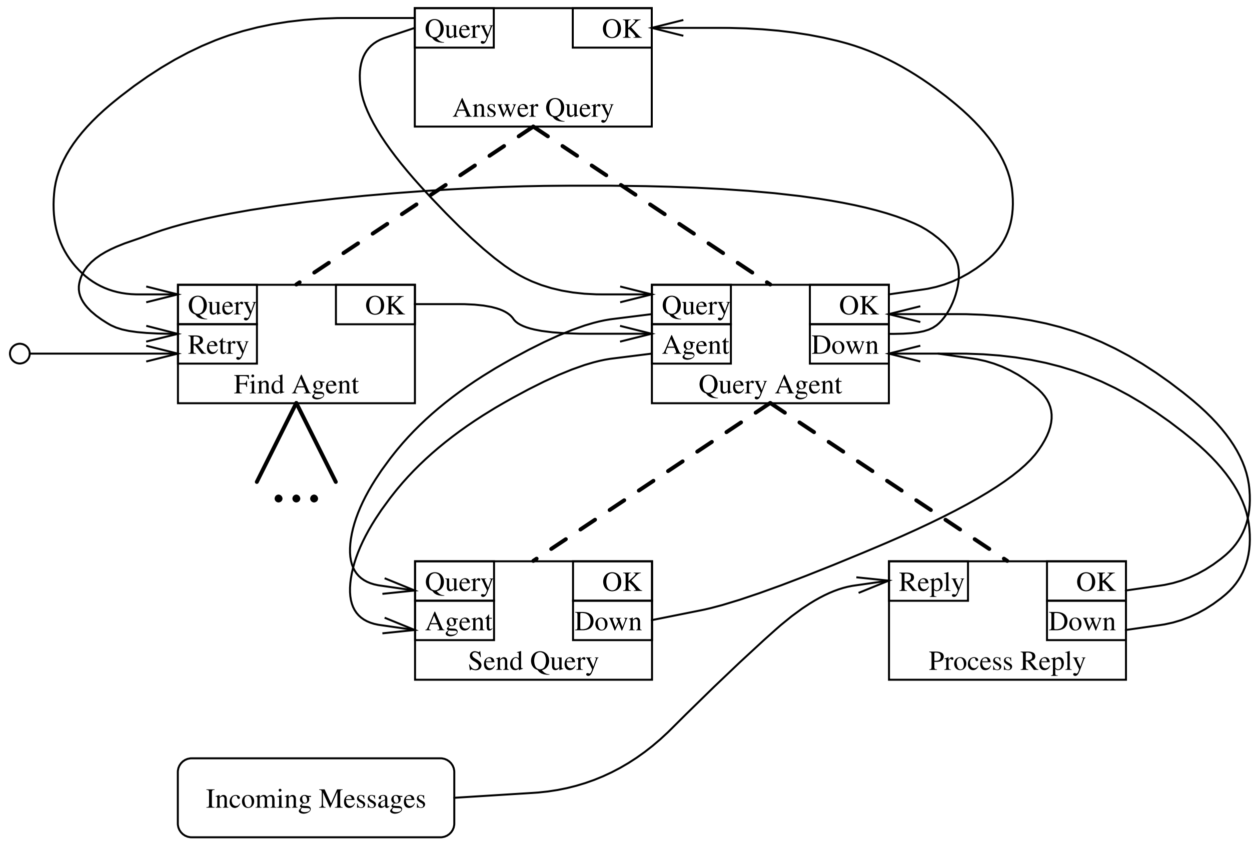

Figure 3: A robust query-answering task structure.

send the query to the other agent and process the reply when it arrives.[8] If the chosen agent is unable to answer the query, a "DOWN" outcome will be propagated upwards to "Query Agent," thereby reactivating the "Find Agent" task. This task structure is robust to the failure or absence of the agent to whom the query is being directed. It will continue finding agents and sending queries until it receives a reply. (A further refinement would be to provide a timeout mechanism, allowing the task to generate a "FAIL" outcome if the query cannot be answered within a specified interval.)

This approach could be applied to other instances of the problem as well. For example, a portfolio agent needing to monitor the price of a stock could use a complex task structure that selects a ticker agent and initiates price monitoring, but also periodically scans for the existence of new ticker agents that could provide better service.

Discussion This approach—encoding contingencies in the task structure—can be used to create behaviors that are quite robust to changes in the environment or organization. Its advantage is that it allows complex behaviors to be generated with a relatively simple planning algorithm and agent architecture. The disadvantages, however, are clear. Conceptually

[8]Incoming messages are dispatched by their KQML `:in-reply-to` field, so that only the response to the query will be sent to the "Process Reply" action. The scheduler will not enable the action until a reply is provided.

simple tasks (such as answering a query) are complicated by the addition of contingencies for occurrences that may be very unlikely. A heavy burden is placed on whoever engineers the task reduction schemas to make *every* task robust, and a good deal of redundant effort may be expended as similar contingencies are encoded in different task reductions. Finally, the reductions can only be as good as the ability of the engineer to anticipate possible failures.

4.2 Monitoring in the Agent

The problems discussed in the previous section have motivated us to investigate several other approaches to the problem. A second general approach is to build execution monitoring mechanisms directly into the agent architecture. These mechanisms can appear in each of the planner, scheduler, and executor components. The idea here is to automate the generation of certain typical behaviors, instead of encoding them explicitly in the individual task reductions.

For example, whenever the planner makes a decision that depends on the capabilities of a particular agent, it keeps a record of that decision. It could also initiate a separate task to periodically gather updated information on that agent, or other agents with similar capabilities. If the updated information suggests a change to the earlier decision, the planner can make a transition from the existing plan to a new one, either by terminating the existing one and replanning from scratch, or by modifying the existing task structure.

The scheduler orders primitive actions, may insert idle actions, and may even customize some actions (in the case of anytime or design-to-time computations). It does this using the periods and deadlines of actions, and statistical information about their expected durations. During processing it is possible for the scheduler to realize that an action will fail (to meet its deadline) well before it is actually executed. Similarly, the executor is charged with monitoring the actual execution of a primitive action. Such actions can be given fixed time budgets and fail (or produce sub-optimal quality results) when that time budget is reached.

Although we have yet to explore this approach extensively, we are hopeful that it will be able to produce behaviors covering the most commonly occurring kinds of change (such as the appearance of new agents). We do acknowledge, though, that this approach will have its limitations, and that some tasks will still require task-specific contingencies to be hand-coded in the reduction schemas.

4.3 Monitoring in the Organization

A third approach to the problem is to use *organizational structure* to reduce the dynamism experienced by individual agents in the system. By "organizational structure", we mean the assignment of certain roles and relationships to specific agents or groups of agents. Organizational structure can help shield individual agents from the ramifications of many kinds of change. For example, most human travel agents are associated with a travel *agency*. Customers can usually be serviced by the agency even if some of the individual agents are unavailable.

Our existing matchmaking mechanism is an example of the use of organizational structure. A new agent entering the system formulates an advertisement describing its capabilities, which is sent to a distinguished matchmaker agent.[9] Whenever any agent needs a service to be performed, it sends a query to the matchmaker asking who can provide that service. The matchmaker returns a list of possible candidates, and the agent can decide among them and then establish a direct relationship with the best candidate. The organizational role played by the matchmaker means that individual agents need not maintain knowledge about all the agents in the system.

Although organizational structure can reduce the impact of dynamic change in the system, it cannot do so without support from the individual agents. Each agent must use either explicit task structures or architectural mechanisms that take advantage of the organizational structure. The matchmaker, for example, would be of no use if the other agents didn't send it advertisements or queries. In our current system, both these behaviors are accomplished by specific task reductions that are shared by all agents.

There are many ways that organizational structure could be used to help agents cope with change. A good example is the use of *brokers*.[10] Recall the earlier example of the portfolio agent that is faced with an ongoing decision of which stock ticker agent to use. A simple organizational solution to this problem would be to use a stock ticker broker (Figure 4). The stock ticker broker would keep track of all existing stock ticker agents and their current capabilities. The portfolio agent would only deal with (in fact, would only know about) the broker, from whom it would obtain all stock price monitoring services. The individual ticker agents could come, go, and change at any time, without any impact on the portfolio agent.[11] Note that the organizational structure doesn't really *eliminate* the problem, since the broker agent is now faced with exactly the same difficulties that the portfolio agent was before. Instead, it *encapsulates* the problem, so that both ahead-of-time design effort and run-time computational effort need not be replicated.

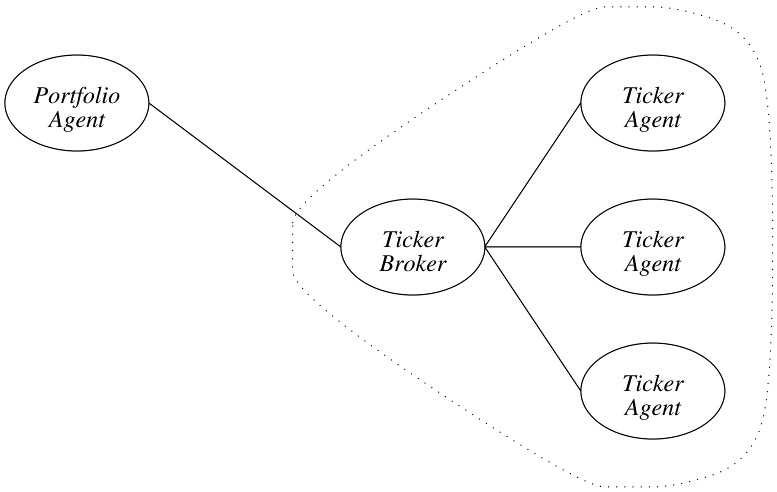

Figure 4: The role of a stock ticker broker agent.

Rigid organizational structures such as this one offer a number of tradeoffs. One pos-

[9]For robustness and scalability, the matchmaking duties could be distributed among many agents, instead of centralized in one.

[10]Brokers are very similar to *facilitators* [7].

[11]Assuming there is always at least one ticker agent able to meet the portfolio agent's needs.

itive attribute is that such structures allow for load balancing and other efficient uses of network resources. By using multiple independent ticker agents, the ticker broker can offer predictable, increased reliability (without requiring every agent in the system to repeat the accompanying calculations). The broker can even provide some simple service integration (e.g., EPS data from one ticker, ex-dividend date from another). While the existence of the broker greatly simplifies life for the clients, there are many potential disadvantages. The broker introduces additional communications latency between the clients and the service providers. The broker constitutes a bottleneck and a single point of failure. These problems can be reduced by using multiple, distributed brokers, but at the cost of increased system complexity. Alternately, more flexible organizations, given time, can adapt to failures. For example, if the ticker broker goes down, the ticker agents could advertise their services directly until the broker is back on line. System performance would degrade, but not fail completely.

In general, we don't believe that any organizational structure will be appropriate for all situations, and the exploration of different possibilities is a significant part of our research agenda.

5 Conclusion

In this paper, we have described issues which must be confronted by a decision-theoretic agent which executes plans in a cooperative, multi-agent domain. How can such an agent maintain the quality of its plans when the agents with which it interacts are appearing, disappearing, and changing their abilities? We propose that solutions to this problem are to be found by using one or more of three possible approaches to plan monitoring: explicit monitoring actions within plans, monitoring mechanisms in the agent architecture, and monitoring structures in the organization.

References

[1] J Ambros-Ingerson and S. Steel. Integrating planning, execution, and monitoring. In *Proc. 7th Nat. Conf. on A.I.*, pages 735–740, 1988.

[2] K.S. Decker, V.R. Lesser, M.V. Nagendra Prasad, and T. Wagner. MACRON: an architecture for multi-agent cooperative information gathering. In *Proccedings of the CIKM-95 Workshop on Intelligent Information Agents*, Baltimore, MD, 1995.

[3] K. Erol, J. Hendler, and D. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, pages 249–254, June 1994.

[4] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, San Francisco, CA, October 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[5] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management CIKM'94*. ACM Press, November 1994.

[6] R.J. Firby. Task networks for controlling continuous processes. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, 1994.

[7] M.R. Genesereth and S.P. Katchpel. Software agents. *Communications of the ACM*, 37(7):48–53,147, 1994.

[8] C. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proc. 15th Int. Joint Conf. on A.I.*, pages 1686–1693, 1995.

[9] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

[10] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. Submitted to AAAI-96 workshop on Theories of Planning, Action, and Control, 1996.