☐ ACHIEVING FLEXIBLE AUTONOMY IN MULTIAGENT SYSTEMS USING CONSTRAINTS

MARK EVANS
JOHN ANDERSON
GEOFF CRYSDALE
Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada R3T 2N2

Organizations influence many aspects of our lives. They exist for one reason: they can accomplish things that individuals cannot. While recent work in high-autonomy systems has shown that autonomy is a critical issue in artificial intelligence (AI) systems, these systems must also be able to cooperate with and rely on one another to deal with complex problems. The autonomy of such systems must be flexible, *in order that agents may solve problems on their own as well as in groups. We have developed a model of distributed problem solving in which coordination of problem-solving agents is viewed as a multiagent constraint-satisfaction planning problem. This paper describes the experimental testbed that we are currently developing to facilitate the investigation of various constraint-based strategies for addressing the coordination issues inherent in cooperative distributed problem-solving domains.*

INTRODUCTION

Management science research has developed many descriptive theories of designing and analyzing human organizations (Galbraith, 1973; March et al., 1958; Simon, 1957). Understanding the behavior of people in organizations has become increasingly important in managing individuals and groups to maximize the performance and effectiveness of an organization. Distributed artificial intelligence (AI) research on the other hand, is concerned with the study and development of organizations of computerized problem solvers (possibly integrated with human problem solvers). The two disciplines share many common issues: how to select organizational structures and decentralized coordination regimes; how to decompose problems and distribute responsibilities and tasks among a number of problem-solving agents; how to coordinate these agents to solve problems cooperatively; and how to measure the effectiveness of the organization in meeting individual and organizational goals (Bond and Gasser, 1988).

It is important, when designing or studying a multiagent system, to consider the *autonomy* of the agents involved. Autonomy, the ability of an agent or system to function independently, is clearly an important characteristic and, unfortunately, one that has been almost universally lacking in AI systems to date (Zeigler, 1990). Completely autonomous AI systems would be desirable in many situations; all too often, humans are expected to intervene where knowledge or senses fail. However, one can think of just as many situations where *completely* autonomous agents would not be desirable. Indeed, the reason humans organize into groups is that more can be accomplished in unison than could be done autonomously. Therefore, in the design of agents for a multiagent system, we require autonomy to be *flexible*, in that agents will sometimes be required to work completely autonomously, but will often be commanded or influenced by others to some degree. This flexibility must also be extensible to higher-level organizations; societies of agents may be autonomous, or dependent on other agents or societies.

We have developed a model of distributed problem solving in which coordination is viewed as a multiagent constraint-satisfaction planning problem (Evans and Anderson, 1989). This architecture supports the basic multiagent planning actions of task decomposition, task distribution, and result integration. It also allows control of agents to be centralized, partially centralized, or completely distributed. This in turn allows agents to be completely autonomous (possibly allowing communication with other agents for information gathering without any formal control or influence, or simply complete ignorance of other agents and their activities), to be completely dependent on other agents, or to possess varying degrees of freedom in between. *Adaptive planning* (the modification of existing plans to cope with execution failure or an unexpected change in the environment) is also supported, through monitoring and replanning using constraint relaxation as a means of negotiation between agents.

This paper describes the experimental testbed that we are currently developing to facilitate the investigation of various constraint-based strategies for addressing the coordination issues inherent in cooperative distributed problem-solving domains. The testbed is being developed in Allegro Common LISP, and consists of a set of constraint-based control and representation mechanisms that enable the construction of cooperative problem-solving agents. Individual agents are implemented as LISP procedures, and tools are provided for adding an intelligent coordination interface to each agent.

We are currently experimenting with these tools by simulating the cooperative activities of a number of agents in a simplified automotive repair shop environment. This domain was chosen because we have found it to be typical of most cooperative problem-solving situations. The operations of a repair shop involve interactions between many classes of agents, which can be divided into two categories: a kernel of agents who provide fundamental problem-solving activities (e.g., mechanics, managers), and a group of peripheral agents who provide ancillary services (e.g., accountants, other repair shops, wholesalers). The organization of these agents is partially hierarchical (e.g., management agents can supervise and dictate the activities of worker agents) but also partially linear (e.g., two mechanics may make decisions as to which tasks each will perform). The autonomy of the agents within the repair shop environment also varies: some work highly autonomously, while others are more tightly controlled. Various examples from this application will be used throughout the paper to illustrate the tools and techniques we are currently investigating; the reader is referred to Evans and Anderson (1989) for additional examples and details of this application. This domain will be stressed considerably in order that the reader may relate the functions provided by the testbed to a real-world scenario. However, it must be emphasized that the testbed is a *flexible* environment, which can accommodate the implementation of many distributed problem-solving situations.

THE EXPERIMENTAL DOMAIN

In order to give the reader a better understanding of the workings of the testbed itself, we will first describe the agents, resources, and flow of information in the domain we have chosen as a test for the system. The following discussion of the automotive repair shop domain is based on analysis of the repair shop associated with a major General Motors dealership in Winnipeg, Manitoba. It is not our intention to imply that all automotive repair shops operate in the manner described; however, we feel that it is representative of many similar organizations and that unique aspects of the operation could be modified to accommodate the peculiarities of other such organizations.

The general layout of the repair shop is illustrated in Fig. 1. The repair shop is divided into several areas including offices, the *tower* (an office overlooking the work area), repair stalls with hoists, and repair stalls without hoists. Each set of repair stalls is outfitted with a limited number of support tools such as diagnostic computers and high-pressure ratchets. Mechanics are required to supply all the other tools they might need.

A repair enters the shop in the form of a work order initiated by a service writer. The work order is routed to the tower operator for task scheduling. Tasks are scheduled on an ad hoc basis: when a mechanic in a given group becomes available, the tower operator assigns the next task from the queue of current work orders associated with that group. There is no ranking of the mechanics within a group; each is considered to have the same abilities. This is not necessarily realistic, however; mechanics aspire to certain designations within their profession (i.e., Grand Master, Master, etc.). These designations are not considered in the task-allocation protocol employed in this particular environment, but they could be represented easily using our model.

The repair shop encompasses several groups of agents, indicated in Fig. 1. The shop manager serves as the shop administrator. He or she is mainly concerned with maintaining public relations and does not need in-depth knowledge of mechanical repairs. The tower operator coordinates work through the shop. He or she assigns tasks to available mechanics, keeps track of the progress of each work order, and arranges the movement of vehicles from stall to stall. The tower operator knows the prescribed duration of each task, and also has a limited amount of mechanical knowledge. The service writers act as liaisons with the customers. They provide the initial description of the repair based on the customer's description of the problem. A service writer must have a relatively good understanding of mechanical repairs in order to transform the customer's description into one that can be used by the mechanics. The service writers also track each work order that they have created as it progresses through the shop. The shop foreman is the head mechanic. He or she has a broad and deep understanding of mechanical repairs, and test drives all vehicles before any work begins and after all work has been completed. The shop foreman also acts as advisor to the other mechanics in problem situations.

Mechanics in the automotive repair shop are divided into several groups whose names are self-explanatory. In addition to the groups shown in Fig. 1, larger groups of mechanics also exist: transmission mechanics and heavy mechanics form a group, as do tune-up and electrical mechanics. Each member of each group can perform all the tasks associated with the group.

The automotive repair shop has an organizational hierarchy that describes the formal authority structure. The shop manager is at the top of this hierarchy and has ultimate authority. Reporting directly to the shop manager are the tower

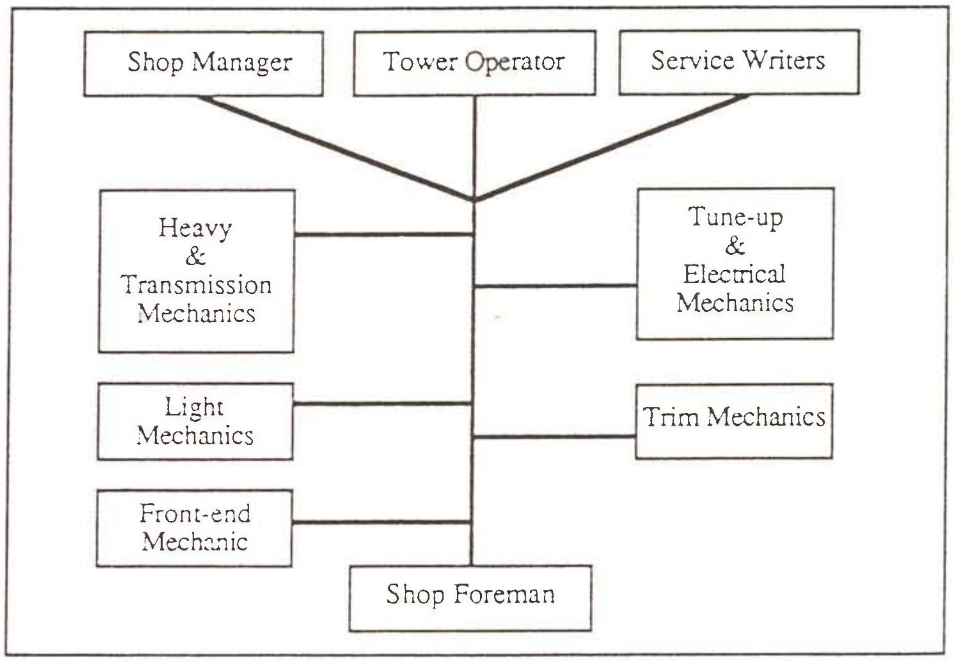


FIGURE 1. The automotive repair shop.

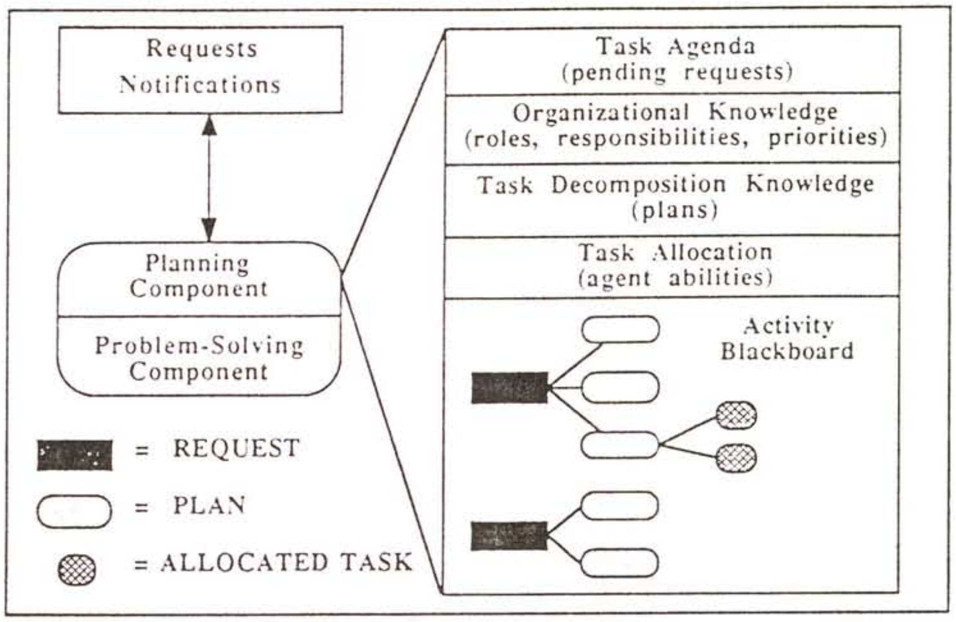

Requests
Notifications
Planning Component
Problem-Solving Component
= REQUEST
= PLAN
= ALLOCATED TASK
Task Agenda (pending requests)
Organizational Knowledge (roles, responsibilities, priorities)
Task Decomposition Knowledge (plans)
Task Allocation (agent abilities)
Activity Blackboard

FIGURE 2. The conceptual model.

operator, the shop foreman, and the service writers. All the mechanics are under the direct supervision of the shop foreman. The shop foreman has the final say on all repairs. The tower operator and the service writers can also exercise a certain amount of control over the mechanics by authorizing repairs.

Established lines of communication that exist within the shop are denoted in Fig. 1. Much of the communication that occurs between the various groups of agents is based on the structure provided by the organizational hierarchy. Among the various groups of mechanics, intragroup communication is quite common in the form of collaboration on difficult repairs; however, intergroup communication is rare.

THE CONCEPTUAL MODEL

The major focus of our research to date has involved the design of control regimes and knowledge representations that allow agents to reason about local activities and cooperatively coordinate global activities with other agents. Agents are implemented as two-component entities, as illustrated in Fig. 2. The *problem-solving* component addresses the tasks that are assigned to the agent; it thus embodies the knowledge required to perform the tasks required of the agent, the inference/control mechanisms necessary to represent that knowledge, and the sensors and effectors necessary for the agent to interact with the outside world. The *planning* component contains a knowledge-based model of the world in which the agent operates; this includes models of the agent's own abilities, the abilities of other agents in the environment, and the relationships between agents. The problem-solving component performs tasks that the agent is able to solve itself, while the planning component acts as an intelligent coordination interface that determines how tasks the agent is to perform may be broken down, (possibly) distributed to other agents, and integrated. Maintaining a distinct coordination interface enables knowledge and control associated with coordination activities to be represented explicitly rather than coded implicitly with basic problem-solving actions. This allows for commonalities among coordination functions to be noted and expressed appropriately, and it also enables development of coordination responsibilities to occur more or less tightly coupled from the problem-solving responsibilities as dictated by the domain.

Agents are grouped into societies (or agencies), and each agent may be a member of many societies simultaneously. Lines of communication exist between agents; these may be either direct or indirect (via another agent), and the communication methods themselves may differ in speed and cost. This information is stored and manipulated by the planning component of each agent. In our current implementation, we are simulating the distribution of agents using a controlled multitasking environment; we can, however, simulate different distribution speeds and costs by modifying an agent's coordination knowledge accordingly.

Agents are also related to one another through lines of *authority*. Authority has many uses in a distributed problem-solving environment (Meehan, 1980). In our model, authority agents perform two main functions. Primarily, they are used to assist in negotiations between agents with opposed interests. Should two agents arrive at a deadlock in negotiations, their respective authority agents (if any) may be consulted for advice on a compromise. Authority agents are also used to provide information of a more globally coherent nature; information of this sort aids in maintaining global plan coherence and the wise allocation of resources.

Societal and authority relationships, together with various constraints that will be described shortly, represent the set of methods available for defining an agent's autonomy. Highly or completely autonomous agents will have few or no lines of authority, and may have only loose societal ties. More tightly controlled agents will have strong societal ties, be strongly influenced by authority agents, and have additional constraints imposed on them.

Organizational Knowledge

Each agent possesses knowledge describing its view of the organization (the view of the organization will vary depending on the agent's role and function: an agent charged with managing a group of subordinate agents will undoubtedly have a more global view of the organization than its subordinates). This knowledge is represented using four different types of knowledge sources (KS's) within each agent. *Plan knowledge* consists of skeletal plans, which prescribe ways of decomposing problems and coordinating the integration of results. *Task knowledge* describes agents (or groups of agents) in the organization that are capable of carrying out tasks specified within plans. *Agent knowledge* describes protocols to be used to interact with other agents when attempting to distribute tasks. *Coordination knowledge* describes an agent's role within the organization, its powers of authority over other agents (or its own authority figures), and resource allocation information. Thus, information about the organization and the agent's means of contributing to it are represented explicitly (the planning process associated with these knowledge sources is described in the section on planning). Much of this knowledge will be incomplete and possibly in conflict with other members of the organization.

This explicit representation of organizational knowledge is not novel. However, the manner in which knowledge is represented and manipulated is innovative: much of the knowledge is represented as sets of *constraints* and corresponding *constraint relaxations*, which can be applied (through negotiation) when conflicts and inconsistencies arise. For example, agent knowledge sources can include constraints such as the information a particular agent requires to perform a specific task or limitations inherent in an agent's ability to perform a task. Coordination regimes defined in our architecture allow each agent to reason about local and global planning by selecting applicable knowledge sources and by satisfying relevant constraints; when conflicts arise, agents can interact cooperatively with other agents to reformulate problem decompositions, task descriptions and distributions, and the integration of results.

Constraints in Multiagent Planning

The representation of a distributed problem-solving environment in terms of constraints yields many advantages. Many of these are at a low level and are provided by the constraint-directed representation. For example, a single representation for all knowledge makes that knowledge more understandable and easily organized; it also simplifies manipulation. More importantly, when a set of constraints cannot be satisfied, selected constraints can be relaxed to arrive at a satisfying solution (the use of relaxation to find a solution that best satisfies a set of constraints has long been known and is the cornerstone of constraint-directed reasoning). Explicit representation of these relaxation methods make for easy selection of alternatives when constraints cannot be satisfied. However, the majority of the benefits of using constraints as a representation mechanism appears at high levels; using the organization presented here, we can flexibly

TABLE 1. Categories of Constraints in Multiagent Planning

Physical	Utility
Maximum weight (hoist class A. 4000)	Resource utility (call-back repair. 8. 10)
Temporal	Resource utility (Muffler repair. 1. 5)
Do before (body repair. engine repair)	Commitment
Availability	Resource commitment (clutch repair. 4. M)
Must have (diagnosis. diagnostic computer)	Communication
Ability	Must inform (additional repair. service writer)
Can perform (tune-up mechanic. light tune-up. 7. 10)	Organizational
	Role (agent 1. tune-up mechanic. 10)
Can perform (light mechanic. tune-up. 0. 0)	Maximize (customer satisfaction)
Relational	Minimize. (Idle resources)
group-task (Engine repair. Transmission repair)	

represent entire organizations and the knowledge within each agent in terms of constraints and relaxations.

The types of constraints necessary for distributed problem solving can be divided into a number of categories, but fall into two broad groups based on their use. Fox and Smith (1984) describe a similar categorization for constraint-based scheduling. However, while these types of constraints are also required in multiagent planning, the differences in problem structure introduce a number of additional types of information that are ideally represented as constraints (Evans and Anderson, 1990b). Thus, while some entirely new types of constraints useful in planning and coordinating cooperative problem-solving activities have emerged, others in the list have been adapted from Fox's work (Fox and Smith, 1984).

The various constraint types are shown in Table 1, along with simple examples of each type of constraint taken from the experimental domain. One group consists of those constraints that influence the development of multiagent plans, while the other group consists of those that influence the coordination of multagent actions. Since, in distributed problem solving, we wish to maintain a balance between an agent's autonomy (i.e., its ability to solve problems on its own and make its own decisions) and its cooperation with other agents (i.e., its ability to consider and balance information from other agents in making decisions), it is crucial that an agent's individual problem-solving abilities be kept distinct from its distributed problem-solving functions. Through the distinctions made in the conceptual model and the distinctions made here in the low-level constraint representation, this is accomplished.

When constructing a plan, one must consider the natural limitations of resources, requiring *physical* constraints. *Temporal* constraints are also crucial in the construction of plans, in order to establish specific orderings among individual requests, messages, and tasks. Given that we require resources in the form of tools or agents to perform our plan, *availability* constraints are also required.

In addition to these, *ability* constraints associate a level of capacity with physical resources and a level of skill with knowledge resources. Ability constraints are much easier to define for physical resources than for knowledge resources. Abilities of physical resources are generally of a predicate nature: either they can do something or they can't. This is illustrated in Table 1: A tune-up mechanic can perform a light tune-up with an ability factor between 7 and 10. Finally, *relational* constraints specify connections among resources and tasks that restrict or enhance actions. Some tasks may be mutually exclusive while others may be complementary. As an example, the same mechanic can perform engine repairs and transmission repairs; therefore, these tasks should be performed consecutively in the same stall.

The types of constraints required to govern interaction between agents are generally of a more abstract nature than are plan-generation constraints. Typically, some plans or specific plan components carry more or less significance than others. Consequently, it is often necessary or desirable to constrain the types of resources that can or should be used to carry out planned actions. We have identified *utility* and *commitment* constraints, which address this facet of task distribution and resource allocation.

Utility constraints govern the selection of planning resources. These constraints dictate the degree of applicability (as indicated by each resource's ability constraints) that must exist between a resource (physical or knowledge) and a task before a resource can be considered a candidate to perform the task. For example, a call-back repair is an important task and is assigned a very high utility constraint to indicate that it requires skilled resources to be used.

Commitment constraints are used to determine the relative importance that a task carries in relation to other tasks being executed or whose execution is pending, locally or globally, within the environment. These constraints set the minimum and maximum amount of effort that should be expended while executing a task. Commitment constraints are also used to determine the amount of effort that should be expended when attempting to resolve incompatibilities that may exist between the distributor and recipient of a task (e.g., availability conflicts, ability conflicts, etc.). For example, in our experimental domain, we can assign a 4-hour clutch repair a medium commitment constraint, to indicate that it is important enough to warrant additional effort to resolve conflicts should they arise. On the other hand, a low commitment constraint would indicate that only minor conflicts should be tolerated. Agents may also become more committed to tasks over time: as an agent expends more and more energy to complete a certain task, its commitment to achieving that task increases as a result of the investment in resources. Thus, an agent's existing plans serve as constraints on its future plans (Bratman et al., 1988), and they force commitment constraints to be modifiable over time as the agent's workload changes and it becomes more or less committed to specific tasks.

While utility and commitment constraints are the most crucial of the coordination constraints in that they are always required, other important coordination constraints exist as well. *Communication* constraints restrict the type and amount of information an agent may share with another agent. Reducing the use of communication resources by being more selective about the messages that are exchanged is accepted as one of the major goals of cooperating agents (Durfee et al., 1989). Communication restrictions can be used to force an agent to interchange only very important information (such as information about agent outages) when the communication network is particularly busy. These constraints can be relaxed to allow the exchange of less vital information (such as agent workload reports), which would allow processing across the organization to be optimized. Constraints also exist with regard to the *types* of information that agents can communicate. For instance, a service writer must be informed if a vehicle requires additional repairs. Furthermore, communication constraints may also identify the *methods* of communication that agents may use. These constraints may be in the form of network protocols for computational agents or real-world constraints for other situations; for example, a mechanic may not communicate with a customer by telephone. However, some of these may also be considered physical constraints since they represent a portion of an agent's physical structure (e.g., a network protocol).

There are many constraints that do not fall into any of the categories discussed or that bridge several categories. We refer to these as *organizational* constraints since they constrain an agent's role and function within an organization. These constraints are of a general nature, identifying such things as global goals (related to the purpose of the organization as a whole: this is loosely analogous to metaplanning in a single-agent system) and their relative importance, the types of interagent activities that should be promoted (and those that should be avoided), levels of authority that one agent has over another, and group norms and beliefs.

Agent Interaction

Agents coordinate cooperative problem solving through the exchange of information with one another. We divide this information into two categories: *problem-solving requests* and *notifications.* These categories are analogous to the *request* and *inform* actions described by Cohen and Perrault (1988).

Problem-solving requests are structures that describe an operation (or set of operations) to be performed, an object (or set of objects) to which the operation is to be applied, and possibly an agent (or class of agents) that is to carry out the operation. A request consists of a set of specific attributes and constraints that describes the work to be performed (examples of such requests are shown in a later section). Within a request, constraints may be imposed on individual attributes, among two or more attributes, or on requests as a whole (e.g., a utility constraint might delimit the quality of plans and/or agents that should be used to fulfill the request). The recipient of the request must coordinate appropriate problem solving in order to fill in the request components with acceptable values. The recipient may carry out these actions autonomously or may choose to distribute some of the task to other agents.

Notifications are more general structures that can be used to convey other types of information between agents. These types of information include logistical information such as agent work-load reports (which may influence the division of labor) or notification of nonfunctioning agents (which are circulated when a given agent finds that some other agent is nonresponsive). Notifications may also convey planning information such as *constraint-violation reports* and *relaxation notifications.* Constraint-violation reports are generated when a specific constraint associated with a request is violated. They describe and justify the violation and provide a suitable number of alternatives (relaxations). An agent receiving such a notification may choose a relaxation (or generate a new one), and return a relaxation notification indicating its proposal. Selecting a suitable relaxation method is a complex process, and a complete description of it can be found in Evans and Anderson (1989). This results in a process of negotiation, which continues until a suitable compromise has been reached. Examples of the use of these structures will appear in a later section.

Planning

The organizational knowledge maintained by an agent dictates the planning involved in developing and coordinating cooperative activities. Ideally, one would like to plan all actions in advance and simply execute them in a prescribed order. Indeed, this plan-then-execute approach is the norm in classical planners such as STRIPS (Fikes and Nilsson, 1971). However, in any real-world environment, this approach is useless: we require the ability to achieve results in real time in areas where the environment is changing either spontaneously or as a result of other agents' actions (Anderson, 1990). Requiring results in real time physically limits the time we may spend planning, as does a dynamic world: chances are that by the time a long plan is constructed, some important aspect of the environment will have been altered in such a way that the plan will be invalidated. In order to cope with these restrictions, it is necessary for a planner to merge planning, execution, and monitoring into a single, ongoing process (Ambros-Ingerson, 1987). The planner must be able to interleave planning and the execution of partial plans, and also incorporate a monitoring or sensing component to recognize both successful execution of partial plans and spontaneous changes in the environment. Single-agent systems have been developed that have incorporated this technique (Ambros-Ingerson, 1987; Wilkins, 1988), but it is even more crucial in any attempt at multiagent planning. This is because the problem that the interleaving process was meant to deal with (a spontaneously changing world) is greatly compounded by the effects of other agents on the environment. In a multiagent world, we may have models of other agents and their activities, but we can never be completely certain they will not interfere with our plans. We also have to coordinate the actions of agents to achieve larger goals, which is difficult to accomplish effectively unless an interleaving process is employed. Thus, a combination of planning and monitored execution forms the basis for our multiagent planning architecture.

The Planning Process

An agent's plans are constructed in a hierarchical fashion on an *activity blackboard,* contained within the planning component of each agent (shown in Fig. 2). When requests are received, they are stored on the blackboard and assigned a priority based on coordination knowledge (e.g., in the repair shop, a previous repair job that has resulted in further problems takes priority over a first-time repair). The highest-ranked request is then selected and used to create a root node of a *plan tree* on the blackboard, an example of which is shown in Fig. 2. There may be several plan trees on the blackboard, each corresponding to a request being processed (only one of which will be active at any given moment). Coordination knowledge is used to determine commitment and utility constraints that denote the amount and type of resources (respectively) to be allocated to the request (this process is described more completely in Evans and Anderson, 1989, 1990a). These constraints are combined with those specified in the request itself and attached to the root node's constraint list. The planning component monitors the expenditure of resources, and low-priority requests may occasionally have to be terminated in order to maintain organizational goals. For example, if a very important repair request is sent to a mechanic (e.g., a call-back or a job for an important customer), the mechanic may have to drop or postpone other requests that are pending. Such requests are rejected, appropriate messages are sent to their sources, and all outstanding processing associated with the requests is terminated.

A request is broken down into a number of alternative or conjunctive plans by applying plan-decomposition knowledge: skeletal plans represented in plan KS's. Each plan KS has a precondition component, which constrains its applicability to a particular request class, and an action component, which specifies how to plan the execution of the request, (i.e., plan steps and coordination information needed to fulfill the request). Using hierarchical planning techniques, further plans are applied to break these plan steps down, until eventually low-level tasks are realized. The plan tree is of the AND–OR variety: at any level, we may have not only conjunctive subplans, but also alternative plans. When a portion of a plan has been reduced to the task level, task-allocation

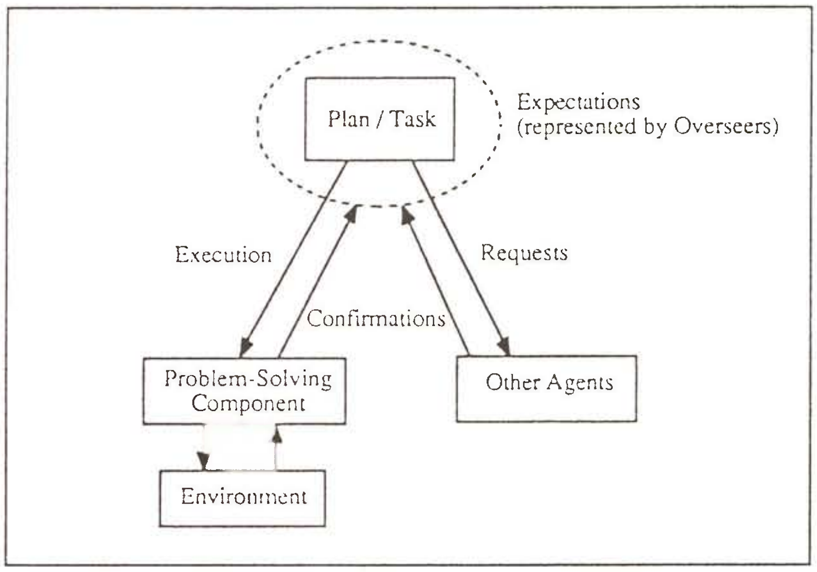

Plan / Task
Expectations (represented by Overseers)
Execution
Requests
Confirmations
Problem-Solving Component
Other Agents
Environment

FIGURE 3. Plan execution and monitoring.

means to rank one KS ahead of another, and thus to prune low-quality or highly incompatible methods and leave a reasonable subset.

Interleaving Planning, Execution, and Monitoring

Interleaving planning with the execution of plans is a difficult task. Our approach to the problem is summarized in Fig. 3. The ability to interleave planning and execution with the architecture's dynamic replanning capabilities is provided through the use of *execution overseers.* These are data structures that are associated with each node (be it a high-level plan node or a low-level task node) in a plan tree and that represent the agent's expectations from that particular node, that is, facts that are expected to change in the environment as a result of the action(s) represented by the plan-tree node. Like the nodes in the plan tree, these will also be hierarchical in nature: a high-level overseer may define expectations such as "Car repaired" or other facts of an abstract nature, while lower-level nodes (under the same basic request) would consist of expectations of various subtasks such as "Remove fan belt" or "Loosen bolt." These expectations are broken down in the same manner in which high-level plans are broken down into low-level plans.

Once a subplan is completely decomposed (i.e., we hit the task level), a task may be executed by our problem-solving component or distributed to another agent. In the latter case, expectation information is included, combined with any constraints that have been agreed upon between the distributing and receiving agents to form a new, high-level execution overseer for the receiving agent. The knowledge, in the form of task and agent knowledge sources, may then be applied to distribute the tasks to the appropriate agent (or to the agent's own problem-solving component) for execution.

Selecting Planning Alternatives

Two major difficulties arise during this planning process: at any given time we have a number of nodes from which to select for expansion and a number of methods for expanding the node. The first problem is dealt with using a combination of a sophisticated scheduler associated with the activity blackboard and the constraint-directed representation detailed earlier. In the same manner as was done with requests, each node in the plan tree is ranked. This ranking is based in part on constraints inherited from higher-level nodes. For example, if a subplan was part of a larger plan to satisfy the important call-back request mentioned earlier, the commitment and utility constraints indicating its importance would be inherited, resulting in a higher rating. Other factors also influence this rating, including the relative importance and difficulty of satisfying other constraints associated with the node. For example, if a very strict time constraint were placed on the action (e.g., it had to be ready for execution in a short time because of the dynamic nature of the domain), this would in turn affect the node's rating. Thus, the most important portions of the most important requests are always worked on first. This sophisticated scheduling approach has been used before in hierarchical, nonlinear planning (Ambros-Ingerson, 1987); however, node-ranking criteria have never been clear. The use of constraints to encode information about the request results in more obvious methods of preferring one node over another. Indeed, constraints are always associated with a ranking of some sort; if we have no method of preferring one constraint over another, the power of the representation is lost.

The second major difficulty is the problem of selecting the most appropriate way of breaking a higher-level plan down into lower-level actions (i.e., choosing which of a repertoire of actions to apply). Here again, the constraint-directed representation helps alleviate the problem. Each applicable KS has a *utility* measure (generated by invoking the KS's ability constraint) indicating the relative utility of the method (i.e., ranking plans, tasks, and agents according to their perceived ability to carry out the required actions); and a *compatibility* measure (generated using constraint satisfaction based on the number and type of invalidated constraints) indicating the degree of compatibility between the activity demanded by the plan node and a KS precondition, ranging from 0 (completely incompatible) to 1 (completely compatible). Each KS also has associated with it a *utility threshold* and a *compatibility threshold*, indicating bare-minimum requirements for the KS to be applicable to a given request (i.e., a method may tolerate some degree of incompatibility, possibly with a reduction in its utility measure to compensate for the incompatibilities that remain). This gives us the major difficulty that occurs when working with conjunctive plans is that subplans may interact with one another. The constraint-directed representation addresses this problem: if a given task interacts with others, the constraints associated with the task will prevent its execution until the affected tasks are also ready for execution, and then the appropriate sequence may be executed (i.e., we impose linearity constraints when necessary).

Given that expectations can be defined, a methodology is also needed to allow expectations to be confirmed or refuted (i.e., we must be able to state whether or not the intended effects of an action were actually achieved). The expectations set by a given plan-tree node may be confirmed in a number of ways, which are also summarized in Fig. 3. If a task has been distributed to another agent, that agent can confirm (via a notification) that the expectations have been met, or if they have not, the agent can indicate what has not been achieved or what constraints have been violated. Expectations of tasks passed to the agent's problem-solving component can be confirmed by sensing equipment in the problem-solving component; for the moment, we follow convention in planning systems (Ambros-Ingerson, 1987; Wilkins, 1988) and simply enter changed predicates via a keyboard to avoid the many problems inherent in perception. Finally, we have expectations at a high level, which are solved by propagating the subplans' satisfied expectations upward to deduce satisfaction at a higher level (part of the process of integrating subplan results to achieve higher-level goals).

As stated earlier, the desire to have realistic execution monitoring is part of the larger goal of implementing interleaved planning and execution. For this, we require two additional facilities. Since we may have alternative subplans that may satisfy a portion of a higher-level goal, we must ensure that, once a portion of one subplan has been expanded enough so that a portion of it may be executed, the other subplan must not be broken down to such a low level (i.e., we don't want to be executing two processes to achieve the same subgoals). This is easily handled by the blackboard architecture along with the constraint-directed representation. When a plan is broken down into alternative subplans, each will be rated based on how well it satisfies the constraints contained in the original plan node. The control strategy will choose the most highly rated alternative. When the chosen subplan is expanded, each of its expanded nodes will be rated as better than the other higher-level subplans. Thus, the alternative subplans will not be expanded unless the other nodes are removed (i.e., the subplan pursued did not succeed). They serve as contingency plans which we may fall back on, but do not interfere with the course of action we have chosen. This technique was originally used in HEARSAY-II to limit hypotheses at high levels of abstraction (Erman et al., 1980) and was first used for this type of opportunistic planning by Hayes-Roth, B., and Hayes-Roth, F. (1988).

The second difficulty with interleaving planning and execution in this environment arises when a partially executed plan is invalidated for some reason. This may be due to an error or alteration in the environment (i.e., unmet expectations) or an inconsistency that arises in an unexpanded, unexecuted portion of the overall plan. In either case, we must cease execution immediately. If there is an error in execution, then we can try repeating the action, backtracking to an alternative plan, or considering other applicable plan-knowledge sources that have not been used.

Of far greater interest from the point of view of multiagent planning are inconsistencies within plans that span multiple agents. These will usually arise due to outside agents' interaction in our plans or to difficulties in coordinating other agents' actions with our own to work cooperatively. The first problem is roughly equivalent to the problem of a dynamic world; we can in many cases ignore the fact that another agent performed some action having a detrimental effect on us and assume that the detrimental effect occurred naturally, rendering the solutions equivalent to those of dealing with execution error and spontaneous effects of nature (Anderson, 1990). The second, however, is a much more difficult problem. Here we must use the constraint-directed representation to allow us to negotiate with other agents to allow for cooperation. We view this negotiation process in terms of selectively relaxing the constraints associated with requests, in order that both agents may reach a satisfactory agreement.

The entire planning process, including task decomposition and allocation as well as interleaving of planning and execution, is summarized in Fig. 4. The

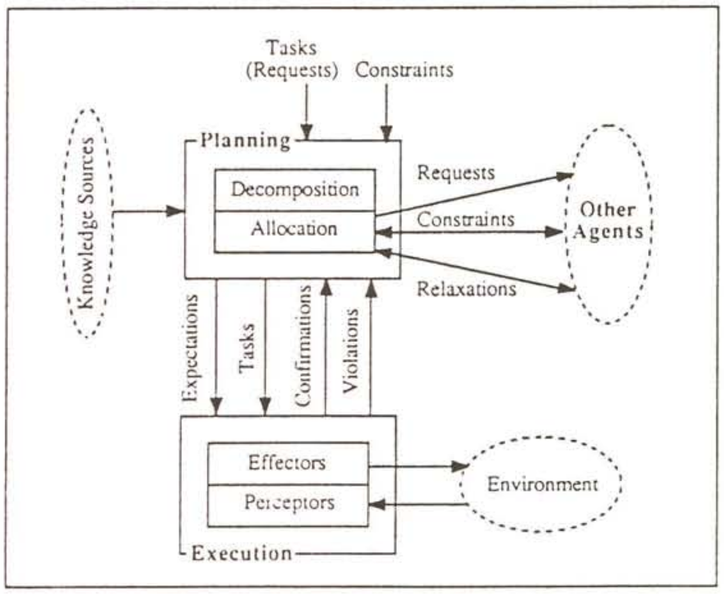


FIGURE 4. The overall planning process.

Request Type:	Service Order
Request ID:	34374
Customer ID:	401303
Task:	Clutch repair, Air conditioning repair
Total Cost:	constraint(<250)
Due Date:	constraint(current time + 8 hours)
	relaxation(if courtesy car is provided then extend Due Date indefinitely)

FIGURE 5. A problem-solving request.

following example from the experimental domain illustrates these concepts (further information may be obtained from Evans and Anderson, 1989, 1990a).

AN EXAMPLE OF THE MULTIAGENT PLANNING PROCESS

To illustrate our problem-solving architecture, we now present a scenario of interaction that is typical of the automotive repair shop domain described earlier. Within this scenario, computerized agents, implemented using our model, simulate the roles of their human counterparts. Once again, while this is the domain we have chosen to implement using the architecture, it is only an example: the architecture is capable of supporting many different styles of coordination. The scenario begins with the following customer's description of a repair:

> I have two problems. When I apply gas as I'm driving along, the car doesn't speed up; the engine just revs. Also, the air conditioner doesn't work.

The service writer translates this repair into a request stating that the clutch slips and that the air conditioner doesn't work. The customer also indicates a willingness to allow the total cost of the repair to reach $250, but requests notification if it will exceed this amount. The customer is also willing to be without the vehicle for a maximum of 1 day unless a courtesy car is provided. Since the estimated clutch repair time is 4 hours and the estimated time for diagnosis of the air conditioning repair is 0.5 hours, a courtesy car will not be issued unless the total time constraint is violated based on feedback received during the actual repairs. The request structure representing this repair, shown in Fig. 5, is routed to the tower operator to be worked into the current shop load.

The tower operator employs his or her task-allocation knowledge to identify the class of agents to which each component of the repair should be assigned. Using this knowledge, the request is placed in the current job schedule for either an electrical mechanic to perform the air conditioning repair or a transmission mechanic to perform the clutch repair. We will assume that the request can be immediately routed to a transmission mechanic to perform the first task of the request: the clutch repair.

Tasks within this request are considered to be totally independent (i.e., the transmission repair cannot affect the electrical repair and vice versa) and, as described earlier, are allocated on an ad hoc basis. In addition, if both mechanics are available, no predetermined ordering is used when the component tasks of the repair are issued. This greatly simplifies the scheduling task and the skill requirements of the tower operator; however, if a higher degree of complexity were required in the planning knowledge used by the tower operator, it could be represented using our model.

The transmission mechanic receives a request from the tower operator to perform the clutch repair. Task-decomposition knowledge is used to identify each of the tasks involved in the repair. Task-allocation knowledge is then employed to identify the agent(s) to whom each of the tasks in the repair should be allocated. This knowledge is represented in the form described earlier, and is displayed abstractly in Fig. 6 (for a syntactical description, see Evans and Anderson, 1989). The task-decomposition knowledge is represented as a set of lower-level plans and tasks that accomplish the job requested. This approach is particularly appropriate for the automotive repair domain, since most mechanics work from this type of "canned" plan. As Fig. 6 illustrates, each individual task is related to a class of agents that are able to perform it. Associated with each of these classes are the specific agents that are known to be candidates for the task. This agent knowledge need not be complete. Alternatively, each agent class is associated with the individual tasks that its members can perform. Again, this knowledge need not be complete. It simply represents a particular agent's view of a certain class of agents (including the class of which it is a member).

The actual process of using the constraints associated with a task to allocate it to the most appropriate agent is illustrated in Fig. 7. Each task will have associated with it a large set of constraints and pointers to other pieces of knowledge relevant to the task (e.g., a description of the task, or knowledge of how to decompose it). Some of the constraints will be associated with the task itself: in Fig. 7, the Adjust-Cable task has such constraints as the qualification of the agent to perform the task, the time it will take, and the tools required. Since tasks (as well as agents) are organized in a hierarchy, constraints will also be inherited from superclasses of tasks (in this case, there will be general policies on all types of transmission repairs, which will further constrain this task). As stated previously, each agent will have a model of the abilities of other agents (and classes of agents). Some of these abilities will be fairly static knowledge (e.g., the agent's strength), while others must be inquired about fairly regularly (e.g., the current status of the agent). These abilities are matched to the constraints dictated by the task and weighted using a heuristic classification process

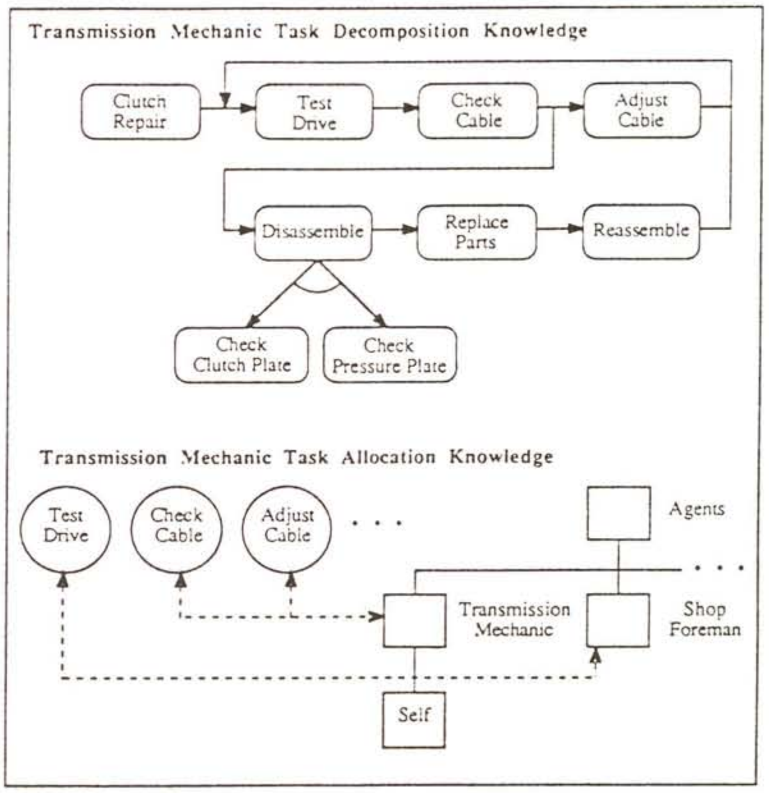


FIGURE 6. Transmission mechanic knowledge.

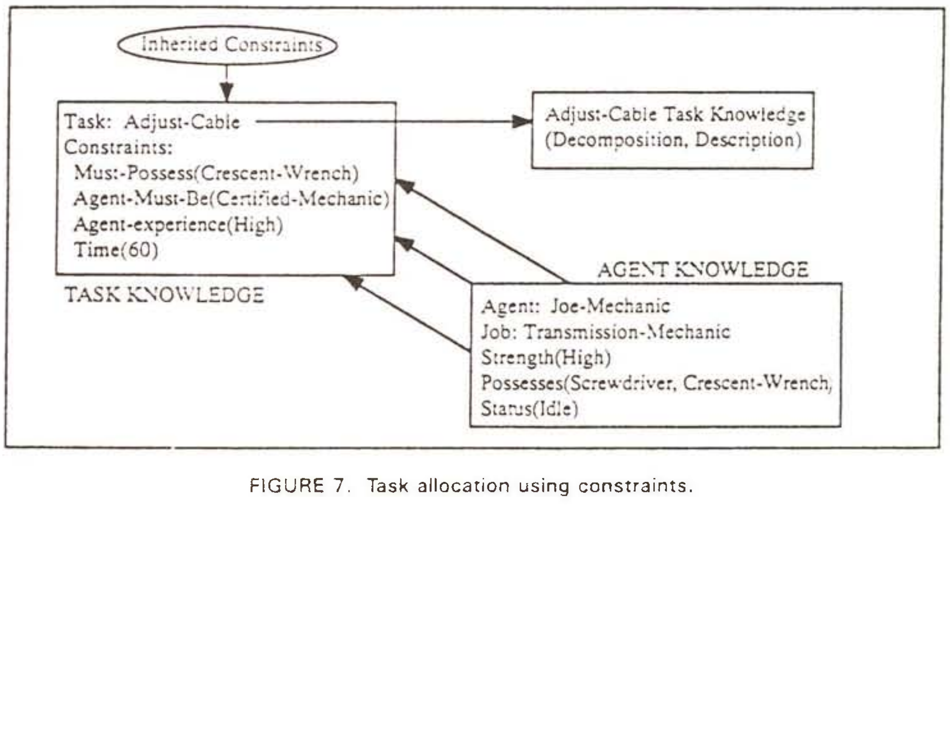

Inherited Constraints
Task: Adjust-Cable
Constraints:
Must-Possess(Crescent-Wrench)
Agent-Must-Be(Certified-Mechanic)
Agent-experience(High)
Time(60)
TASK KNOWLEDGE
Adjust-Cable Task Knowledge
(Decomposition, Description)
AGENT KNOWLEDGE
Agent: Joe-Mechanic
Job: Transmission-Mechanic
Strength(High)
Possesses(Screwdriver, Crescent-Wrench)
Status(Idle)

FIGURE 7. Task allocation using constraints.

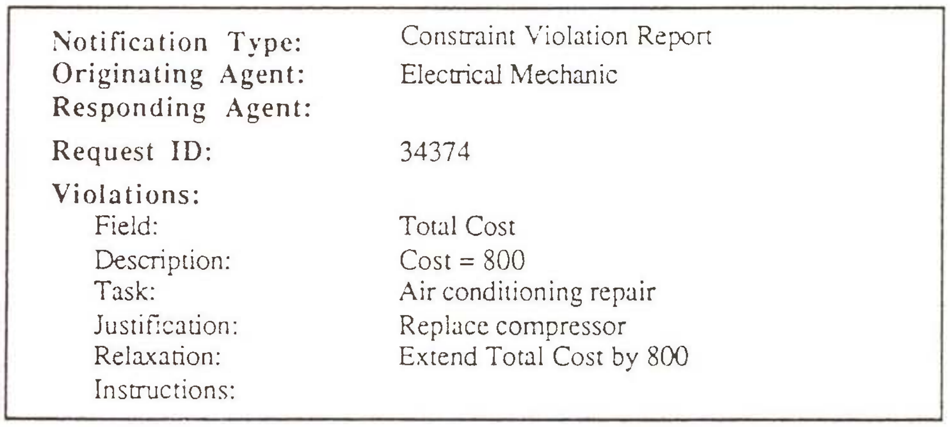
(Evans and Anderson, 1989). Further details of the task-allocation process can be found in Evans and Anderson (1991).

Upon successful completion of the repair, the transmission mechanic sends an appropriate notification to the tower operator, who updates the service order accordingly. The next task to be performed is the air conditioning repair. When an electrical mechanic becomes available and the work order is at the top of the request queue for electrical mechanics, the tower operator issues an appropriate request.

The electrical mechanic will use his or her task-decomposition and task-allocation knowledge in the same way as the transmission mechanic. The mechanic will perform the majority of the tasks, but may send requests to other agents as required. In this case, the electrical mechanic proceeds with the diagnosis and determines that the compressor needs to be replaced. The total-cost constraint, passed on to the electrical mechanic as part of the tower operator's request and updated after the transmission repair, will be violated if the compressor is replaced.

This event causes the electrical mechanic to send a notification, similar to that shown in Fig. 8, to the originating service writer for authorization to continue with the repair (the service writer may then communicate directly with the customer or may authorize the repair based on existing warrantees). Included with the notification is a proposed relaxation of the constraint being violated, proposing that the total-cost constraint be modified to include the cost of replacing the compressor.

Upon receiving the notification from the electrical mechanic, the service writer issues a notification to the customer indicating the constraint violation and the proposed constraint relaxation. Given the cost involved and the nature of the component to be repaired (most people in Winnipeg would consider air condi-



Notification Type:	Constraint Violation Report
Originating Agent:	Electrical Mechanic
Responding Agent:	
Request ID:	34374
Violations:	
Field:	Total Cost
Description:	Cost = 800
Task:	Air conditioning repair
Justification:	Replace compressor
Relaxation:	Extend Total Cost by 800
Instructions:	

FIGURE 8. Constraint-violation report.

Notification Type:	Constraint Violation Report
Originating Agent:	Electrical Mechanic
Responding Agent:	Service Writer
Request ID:	34374
Violations:	
Field:	Total Cost
Description:	Cost = 800
Task:	Air conditioning repair
Justification:	Replace compressor
Relaxation:	Extend Total Cost by 800
Instructions:	Abandon air conditioning repair

FIGURE 9. Updated constraint-violation report.

tioning a luxury item), the customer decides that the repair should not proceed. The proposed relaxation is rejected, and negotiation is terminated. The service writer subsequently issues corresponding notifications (Fig. 9) to the tower operator, so that the original request can be updated accordingly, and to the electrical mechanic, so that the repair will not be completed.

RELATIONSHIP TO HIGH-AUTONOMY SYSTEMS RESEARCH

When we view this architecture from the perspective of high-autonomy systems research, a number of important points emerge. One is that the use of modeling techniques has been extensively explored in the development of high-autonomy systems. However, our approach is unique in that we employ constraint-based representation and control paradigms to capture and manipulate models of potential activities and agent abilities to contribute to these activities. The use of constraints provides a homogeneous and natural means of representing the models that will ultimately drive agent interactions. Our categorization of constraints enables the representation and application of partial models addressing different aspects of multiagent planning.

As has already been stated, agents in a multiagent system require flexible autonomy in order to adapt to a wide range of problem-solving situations. The architecture includes the decision, action, and perception components necessary for high autonomy (Zeigler, 1990), and the use of a constraint-based representation for the various aspects of an agent's relationship to others allows us to relax or constrain these relationships, thus producing flexible autonomy. Our constraint-directed representation is also hierarchical in nature, with constraints at the operational, agent, and organizational levels (for a more complete explanation of this view of constraints, see Evans and Anderson, 1990b). Since an agent's autonomy is defined largely through these constraints, the concept of autonomy itself becomes hierarchical: agents can be subservient or completely autonomous; likewise, groups of agents can be self-sufficient or dependent on others.

While this defines an abstractional hierarchy based on organization for autonomy, autonomy can also be viewed as a hierarchy based on ability. Zeigler (1990) describes three "levels of achievement" for autonomy. The first is characterized by the ability to achieve prespecified objectives using knowledge in the form of models; the second is characterized by the ability to adapt to major environmental changes; and the third is characterized by the ability to develop its own objectives. The constraint-directed architecture described here can be used to define agents (and societies) at all levels of autonomy. Level 1 agents are relatively simplistic, and are very easily definable within this architecture: their objectives arrive in the form of requests, and are broken down and distributed or executed using models of the agent itself and the other agents around it. The perception components and our method of interleaving planning and execution allow level 2 agents to be defined: the environment may change, but existing plans can be repaired by the agent to cope with these changes. Lines of authority, communication, and organizational constraints can also be altered to change the structure of an entire organization in order to adapt to change. As for level 3, the concept of an agent defining its own objectives is somewhat vague. However, depending on the interpretation of this statement, the architecture can also define level 3 agents. The receipt of information from the environment or from other agents may cause an agent to alter its plans (based on its own knowledge about the world) or to opportunistically generate new requests based on that information (i.e., the information may cause the agent to realize that this is an opportune time to perform some action). Whether intentions such as these can be characterized as purely internal or not remains an open question (Bratman, 1987).

From a design perspective, our architecture can be used to test operational designs of systems consisting of highly autonomous agents. Such design procedures require many influential decisions to be made such as the structure of agent groupings, the distribution of authority across agents, potential bottlenecks among agent activities, or alternative activities. Our architecture can be used to create experimental testbeds for examining the ramifications of these decisions before committing to a final system design. Furthermore, the architecture can be used to simulate existing systems to measure effectiveness, uncover bottlenecks, and evaluate streamlining procedures. From an implementation perspective, the architecture can be used to develop working systems consisting of several cooperating knowledge-based or expert systems. Consequently, systems developed from diverse, yet interrelated applications can be integrated to capture the benefits of potential interactions among their individual abilities.

CONCLUSIONS

We have described an architecture that enables the design and development of organizations of autonomous but cooperative agents. This architecture supports the basic multiagent planning actions of task decomposition, task distribution, and result integration through the use of activity and agent modeling. Control among agents may be centralized, partially centralized, or completely distributed depending on the characteristics of the application; the authority exercised among agents can be captured in the agent models and used to bound the influence of one agent over another. The architecture also provides adaptive planning when bottlenecks or incompatibilities arise, through monitoring and replanning using constraint relaxation as a means of negotiation between agents.

We have also demonstrated that much of the knowledge necessary for planning and coordinating cooperative activities among multiple problem-solving agents can be expressed in terms of constraints. Although (as stated earlier) constraints have long been recognized as an essential component in planning, their representation power in the coordination of multiple problem-solving agents has yet to be fully appreciated. The representation of a distributed problem-solving environment in terms of constraints yields many advantages. Some of these are at a low level. For example, a *single* representation for all knowledge makes knowledge more understandable and easily organized; it also makes manipulation of that knowledge easier. Explicit representation of relaxation methods makes for easy selection of alternatives when constraints cannot be satisfied. However, the majority of the benefits of using constraints as a representation mechanism appear at high levels; using our model, we can represent entire organizations and the knowledge within each agent in terms of constraints and relaxations.

Throughout this paper, we have examined the application of our work to the activities and agents in an automotive repair shop. The architecture itself, however, is generic in nature. In effect, the development of a system involves creating N individual agents with appropriate local problem-solving abilities (akin to N autonomous knowledge-based systems) and then developing coordination interfaces for each agent, which involves specifying plan and agent knowledge (activity and agent models) of the agent itself and of *some* of the other *related* agents in the system. The constraint-directed control mechanism provided in the architecture can then interpret these models and construct and coordinate appropriate agent interactions to achieve a synergy of agent abilities within the system.

Research on this project is currently continuing in a number of different directions. First, an implementation in cooperation with the Department of Electrical Engineering at the University of Manitoba is under way, using this constraint-directed model to control robotic agents (Evans et al., 1991). The constraint-directed reasoning methods presented here are also being integrated into a system for combining reactive and strategic planning within a single agent (Anderson and Evans, 1991a, 1991b).

REFERENCES

Ambros-Ingerson, J. 1987. IPEM: Integrated Planning, Execution and Monitoring. Unpublished M. Phil. dissertation, Computer Science, University of Essex.

Anderson, J. 1990. Toward a Theory of Temporal Reasoning for Practical Planning Systems. Technical Report, Department of Computer Science, University of Manitoba, January.

Anderson, J., and Evans, M. 1991a. An Architecture for Reactive and Strategic Planning. *Proc. 4th University of New Brunswick Artificial Intelligence Symposium*, September, Fredricton, pp. 195–210.

Anderson, J., and Evans, M. 1991b. What to Do Next: Using Constraints in Reactive Planning. Working Paper, Department of Computer Science, University of Manitoba.

Bond, A., and Gasser, L. 1988. *Readings in Distributed Artificial Intelligence.* San Mateo, Calif.: Morgan Kaufmann.

Bratman, M. 1987. *Intentions, Plans, and Practical Reason.* Cambridge, Mass.: Harvad.

Bratman, M., Israel, D., and Pollack, M. 1988. Plans and Resource-Bounded Practical Reasoning. Technical Note 425R, SRI International.

Cohen, P., and Perrault, C. 1988. Elements of a plan-based theory of speech acts. In *Readings in Distributed Artificial Intelligence*, eds. A. Bond and L. Gasser, pp. 169–186. San Mateo, Calif.: Morgan Kaufmann.

Durfee, E., Lesser, V., and Corkill, D. 1989. Trends in cooperative distributed problem solving. *IEEE Trans. Knowl. Data Eng.*, 1(1):63–83.

Erman, L., Hayes-Roth, F., Lesser, V., and Reddy, D. R. 1980. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Comput. Surv.* 12(2):213–253.

Evans, M., and Anderson, J. 1989. A constraint-directed architecture for multi-agent planning. *Proc. 9th AAAI Distributed Artificial Intelligence Workshop*, pp. 1–24, Seattle.

Evans, M., and Anderson, J. 1990a. Constraint-directed intelligent control in multi-agent problem solving. In *AI, Simulation, and Planning in High Autonomy Systems*, eds. B. Zeigler and J. Rozenblit, pp. 42-50. Los Almitos, Calif.: IEEE Computer Society Press.

Evans, M., and Anderson, J. 1990b. An Analysis of Constraints for Multi-Agent Problem Solving. Technical Report, Department of Computer Science, University of Manitoba, February.

Evans, M., and Anderson, J. 1991. Flexible task allocation in heterogeneous cooperative systems. *AAAI Workshop on Heterogeneous Cooperative Systems*, July, pp. 14–26.

Evans, M., Jagannathan, S., and Anderson, J. 1991. Using Constraint-Directed Reasoning in the Control of Autonomous Robots. Working paper, Departments of Computer Science and Electrical Engineering, University of Manitoba.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2:189–208.

Fox, M., and Smith, S. 1984. ISIS: A knowledge-based system for factory scheduling. *Int. J. Exp. Syst.* 1(1):25–46.

Galbraith, J., 1973. *Designing Complex Organizations.* Reading, Mass.: Addison-Wesley.

Hayes-Roth, B., and Hayes-Roth, F. 1988. A cognitive model of planning. In *Readings in Cognitive Science*, eds. A. Collins and E. E. Smith, pp. 496–513. San Mateo, Calif.: Morgan Kaufmann.

March, J., Simon, H., and Guetzkow, H. 1958. *Organizations.* New York: Wiley.

Meehan, J. 1980. Everything you ever wanted to know about authority structures but were unable to represent. *National Conference on Artificial Intelligence*, pp. 212–214. Stanford, Calif.

Simon, H. 1957. *Models of M:n.* New York: Wiley.

Wilkins, D. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* San Mateo, Calif.: Morgan Kaufmann.

Zeigler, B. 1990. High autonomy systems: Concepts and models. In *AI, Simulation, and Planning in High Autonomy Systems*, eds. B. Zeigler and J. Rozenblit, pp. 2–7. Los Almitos, Calif.: IEEE Computer Society Press.