A FRAMEWORK FOR DISTRIBUTED PROBLEM SOLVING

Dave McArthur, Randy Steeb and Stephanie Cammarata

The Rand Corporation
1700 Main St.
Santa Monica, CA, 90406.

ABSTRACT

Situations in which several agents must interact to achieve goals present difficulties of coordination and cooperation not found in single-agent problem solving contexts. Techniques for coordination and cooperation required in group problem solving are not well understood because most AI models deal with cases in which problems are solved by a single agent. In this paper we present a framework for distributed problem solving that describes some of the expertise an agent working in an multi-agent environment must have. An application of the framework to the domain of air-traffic control is discussed. Here each aircraft is viewed as an agent that must cooperate with others to achieve a conflict-free plan.

I INTRODUCTION

Distributed Artificial Intelligence (DAI) is concerned with problem solving in which several agents interact to achieve goals. An agent who is part of a group attempting to solve a problem in a distributed fashion must exhibit several abilities or competences, for example:

1. The agent must be able to continuously update possibly incomplete or incorrect world models (caused by limited situation assessment or by the existence of agents that change the environment dynamically and unpredictably).

2. The agent must be able to integrate information coming asynchronously from other agents.

3. The agent must know when new information invalidates his current attempt to plan and be able to modify his plan during execution if new and inconsistent information is uncovered.

4. The agent must be able to interrupt his planning to help others acquire knowledge to achieve their goals (especially when his goals and theirs interact or conflict).

5. The agent must be able to get others to help satisfy his goal or play his role.

Not all these competences are required of single-agent problem solvers, and, since most existing models of problem solving (e.g., STRIPS [1], BUILD [2], NOAH [3] and many others) focus on single-agent problems, little is known about how agents solving distributed problems achieve such competences. We have been attempting to understand distributed problem solving by developing computational models of agents that have these abilities. In this paper we present a view of agents in group problem solving situations that we have developed, and discuss a specific problem solver for distributed air-traffic control (DATC) consistent with this view.

II A VIEW OF DISTRIBUTED PROBLEM SOLVING

Using competences such as those discussed above, we have evolved a view of the important features of distributed problem solving agents that differs considerably from the view traditionally adopted in models of single-agent problem solvers. For example, many theories [1], [2] suggest that a central feature of the single-agent problem solver, is that its activities are decomposed into separate, strictly ordered, phases of information gathering, planning, and execution. However, the above competences indicate that, in situations where a given agent is not the sole cause of change, and therefore where not all important consequences of a planned action can be foreseen at the time of planning, it is essential that the agent be able to effectively interweave information gathering, planning and execution tasks.

Briefly, the main tenets of our view are:

1. Each agent has several distinct kinds of generic tasks such as information gathering (sensing and input communication), information distribution (output communication), planning, plan evaluation, plan fixing, and plan execution.

2. Each kind of generic task invocation (or task instance) is a process: it can be suspended and resumed; hence tasks can be interwoven without losing continuity.

3. Each agent has a knowledge-base that represents his beliefs about other agents and their intentions, as well as information about the static environment and his own intentions.

4. Within a single-agent, the knowledge-base is shared by all task instances, like a HEARSAY blackboard [4]. Any change in the knowledge-base made by a task (e.g., information gathering) while another task is suspended (e.g., planning) will be visible to the latter

when it resumes. Thus tasks such as planning exhibit currency as well as continuity: they do not base computations on an outdated world-model.

5. Task instances are both data-driven and event-driven. Instances of generic tasks are triggered in two ways: by sets of well-defined knowledge-base states; or well-defined events which result in changes to the knowledge-base. Tasks that are created do not immediately get executed but are enabled and may compete for processing resources.

6. Each enabled task has a limited amount of self-knowledge, including explicit intentions and validity conditions. This information can be used to determine if a task is justified in continuing as conditions change. Thus tasks will exhibit relevance.

7. Enabled tasks are not invoked in a fixed order, as in single-agent problem solvers. Rather the agent acts as a scheduler, reasoning about the ordering of task instances. More specifically, the agent uses a set of heuristic rules to prioritize processes representing enabled tasks.

8. A task selected by the agent for execution is not necessarily allowed to run to completion. It is given a quantum of processing resources (time). The size of this quantum is also controlled by the agent.

9. During the execution of a task or process (i) the task may complete, in which case it is eliminated from the set competing for resources; (ii) new tasks may be created because of knowledge-base changes effected by the running task (iii) the changes may cause existing tasks to lose their justification.

10. After a task has consumed its allocated supply of resources (i.e., time), the agent reorders the priority of enabled tasks and selects a new one to run, in light of the conditions in the altered knowledge-base. It also eliminates unjustified tasks (if the tasks have not eliminated themselves).

11. This procedure iterates until there are no more enabled tasks worth running.

Speaking generally, then, we view the agent in a group problem solving situation as a kind of knowledge-based operating system. The view is similar to a HEARSAY blackboard model [4], except that (i) tasks are not suspended and resumed in such a model, (ii) in a HEARSAY-like model a given KS can have several different simultaneous activations, while our framework permits only one instance of a generic at a time, and (iii) the heuristic rules used by agents in our framework (to control the selection of tasks to run) typically use more knowledge of global context than do the scheduling rules in HEARSAY (which control KS invocation). Hence, the performance of our agents may often appear more goal-directed and less data-directed than comparable HEARSAY agents [5].

Our view is not a model of an agent in a specific distributed domain, but rather represents a theoretical framework for describing distributed agents or a set of guidelines for constructing a specific model. Adhering to the framework, the user still needs to provide several sorts of domain-specific expertise. These include the procedures that comprise each generic task, the triggering conditions under which a task instance is to be created, the validity conditions under which it is permitted to continue, and the heuristic rules that order the priority of enabled tasks in light of the current state of knowledge.

In order to facilitate the development of our specific distributed problem solvers, we have implemented the framework in a simple task language. The task language is a set of INTERLISP functions that provides the user with a convenient vocabulary stating the required domain-specific expertise. Once stated, the task language takes care of all the specifics of task management. It insures that an appropriate task instance is enabled whenever the triggering conditions of a user-defined generic task are met. The task language also takes care of the low-level implementation of tasks as resumable coroutines, and guarantees that these processes suspend after consuming the appropriate amount of time. Finally, it handles the details of scheduling the next task to run; the user only needs to state the properties of the scheduler his application requires. (For more details on the capabilities of the task language, see [6]). By attending to the details of task creation and management, the task language frees the user to focus on the theoretically more interesting issues of designing (and debugging) rules that achieve the appropriate interweaving of tasks.

To get a more concrete idea of the value of our view of distributed problem solving agents and of how the user interacts with the task language to create a particular agent, we will now discuss examples of use of the language in developing our distributed air-traffic control (DATC) system. We begin with a brief description of the air-traffic control domain.

III DISTRIBUTED AIR-TRAFFIC CONTROL

The domain of Air Traffic Control exhibits a number of features that make it useful for studying distributed problem solving. Our ATC system consists of several functionally identical agents, one associated with each aircraft in a rectangular (14 x 23 mile) airspace. The agents (aircraft) essentially operate in parallel. (The details involved in getting the agents to time-share on a single processor are invisible to the user). Aircraft may enter the airspace at any time, at any one of 9 infixes on the borders of the airspace, or from one of two airports. The main goal of each aircraft is to traverse the airspace to an assigned destination--either a boundary outfix, or an airport. Each aircraft has only a limited sensory horizon, hence its knowledge of the world is never complete and it must continually gather information as it moves through the airspace. Information may

be accumulated either by sensing or communication. Agents are allowed to communicate over a limited band-width channel to other aircraft, for purposes of exchanging information and instructions.

DATC is a group problem not only because agents may help one another gather information but also because the goals of one agent may interact with those of another. Goal interactions come in the form of shared conflicts. A conflict between two agents arises when, according to their current plans, the two will violate minimum separation requirements at some point in the future. (Currently we require a separation of 3 miles and 1000 feet of altitude). When shared conflicts arise, agents must negotiate to solve them. In a crowded airspace, such goal conflicts can get particularly complex, and involve several aircraft, thus necessitating a high degree of group cooperation. Our goal has been to discover problem solving methods by which a DATC agent can eliminate shared conflicts.

To define our system within the framework of the task language we must identify the tasks comprising each agent and specify the expertise associated with each task. The top-level generic tasks of each DATC agent currently include:

1. Sensing (gathering information about positions and types of other aircraft).

2. Input-communication (gathering information about routes, plans and requests of other aircraft).

3. Output-communication (distributing information about your routes, plans, and requests to others).

4. Initial plan generation (computing a reasonable path through the airspace to one's outfix).

5. Plan evaluation (finding conflicts between your plan and the plans you believe others are following; reviewing new information for consistency with beliefs about others' plans).

6. Plan fixing (using existing plans and evaluations to create new plans that avoid conflicts with others).

7. Plan execution.

A. Defining DATC generic tasks and invocation conditions

A major part of defining a generic task is stipulating the conditions under which an instance of a generic task should be created. Consider plan evaluation. We want to define the DATC agent so that an evaluation task is created when (i) the agent has a plan and, via some information gathering task, learns the plan of some other aircraft, (ii) the agent changes his own plan, or (iii) the agent believes he knows the plan of another aircraft and senses a new position for that aircraft that may not be consistent with what the believed plan predicts. In the first two cases the kind of evaluation needed is "conflict detection"; in the third it is "consistency checking". Using the task language, the "conflict detection" case is implemented as follows:

```
(1) (CREATE-SUBTASK-TYPE 'Evaluation 'Agent)
(2) (CREATE-SUBTASK-TYPE 'DetectConflict
                         'Evaluation)
(3) (SET-TASK-FUNCALL 'DetectConflict
      '(COMPUTE-CONFLICTS Aircraft Other))
(4) (DEFINE-TASK-TRIGGER
      'DetectConflict
      'Evaluation
      '(SET-AIRCRAFT-PLAN Other <newplan>)
      '(Check new plan of Other for conflicts
        against yours)
      '(AND (AIRCRAFT-PLAN Other)
            (EQUAL <newplan>
                   (AIRCRAFT-PLAN Other))))
```

(1) Establishes the generic task of plan evaluation. Evaluation can be thought of as a class object in the SMALLTALK sense [7]. Instances of Evaluation represent specific plan evaluation tasks that might be created. The second argument in (1) says that when a plan evaluation task is created it is to be a a top-level task of the agent.

(2) establishes a generic subtask of plan evaluation. When triggering conditions of DetectConflict are met and an instance of it is created, the instance becomes a subtask of the current Evaluation task of the agent. Thus while the agent is a scheduler that chooses from among enabled tasks that are instances of generics such as Evaluation and Sensing, an Evaluation instance itself is a scheduler that chooses from among instances of CheckConsistency and DetectConflict.

(3) associates a function call with DetectConflict. When an instance of a generic task becomes enabled, it may be selected to execute by the Evaluation task. If the task has previously executed and suspended, Evaluation knows where to resume; if this is the first time the task has been allocated processing resources, Evaluation needs to have a way of initiating the task. It does this by evaluating the function call. Note (3) presupposes COMPUTE-CONFLICTS has been defined by the user and encodes the appropriate expertise.

(4) stipulates the conditions under which task instances of DetectConflict will be created and become a subtask of Evaluation. Roughly, it says "Any time you believe you know some other aircraft's plan, it is reasonable to create a DetectConflict task as a subtask of the current Evaluation task, to see if your current plan conflicts with his new one. This task is justified as long as you still believe you know the aircraft's plan and it is the new one".

B. Defining rules that interweave DATC task instances

Declarations such as (4) show how task creation is data-driven, how tasks insure that they are relevant as conditions change, and how tasks may be suspended and resumed. But to interweave tasks such as plan evaluation, information gathering, etc., permitting the DATC agent to perform intelligently,

we still need to define heuristic rules that will order the priority of enabled tasks. Two rules currently used are:

```
(1) (DEFINE-SCHEDULING-RULE 'Agent
      (if (TASK-TYPE Task)='PlanFixing
         and (EXISTS-TASK-OF-TYPE 'Agent
                                  'Evaluation)
        then (SET-TASK-PRIORITY Task 0)))
(2) (DEFINE-SCHEDULING-RULE 'Agent
      (if (TASK-TYPE Task)='SendReplanRequest
         and (EXISTS-TASK-OF-TYPE 'Agent
                                  'PlanFixing)
    and (GREATERP (TASK-TOTAL-TIME
                    (TASK-OF-TYPE 'Agent
                                  'PlanFixing))
                  5000)
    and (NOT (IN-IMMINENT-DANGER Aircraft))
   then (SET-TASK-PRIORITY
           (TASK-OF-TYPE 'Agent 'PlanFixing)
           0)
        (SET-TASK-PRIORITY Task 200)))
```

(1) defines a top-level scheduling rule of a DATC agent; it helps the agent decide which of the enabled top-level tasks to execute next. The rule says that if PlanFixing is enabled (because an aircraft's plan has a conflict in it), then it is a good idea not to allocate further resources to this task if there is some evidence that the conflict-status of the plan should be re-evaluated. The rationale is that the Evaluation task may have been enabled by receipt of a new plan for the aircraft causing the conflict and this plan may avoid the conflict.

(2) also defines a top-level scheduling rule for the DATC agent. Details aside, its role is to decide when a given agent (aircraft) has tried "hard enough" to solve a conflict shared with another aircraft. Note that "hard enough" has a natural definition in terms of the processing resources (time) that have already been devoted to attempts at PlanFixing. If this criterion is met, the agent will use his other option in solving a shared conflict--he will ask the other conflictee to try to resolve it (by invoking the SendReplanRequest task) instead of expending more effort to try to resolve the conflict himself.

Rules such as (1) and (2) are the key to the DATC agent's ability to interweave its several enabled tasks in way that is sensitive to changing conditions. Many of the rules the DATC problem solver currently employs are devoted to ordering tasks purely "internal" to the agent. These tasks, including sensing, evaluation, plan-fixing, and plan execution, often must be interwoven because of the existence of external, unpredictable, agents. However, these tasks do not directly involve those agents. On the other hand, rules like (2) reason about tasks that involve interaction (communication) with others, either in the service of one's own goal or other's goals. In short, these rules encode cooperative strategies, which are critical to any effective group problem solving activity.

IV CONCLUSIONS

We are developing a system for distributed problem solving in DATC using our task language and adhering to the general view of distributed problem solving agents set out above. Our aim is not to produce a fixed distributed ATC problem solver so much as to perform computational experiments to discover policies for interweaving tasks and strategies of cooperation which are sound across a variety of group problem solving conditions.

We believe the development of such policies to be a central goal for DAI. We are pursuing this goal by implementing several alternative policies and strategies, then evaluating the performances of the resulting systems. In order to complete our experiments, therefore, it is necessary to be able to easily modify the strategies and policies embedded in the system. This is straightforward because, by developing the DATC problem solver in the context of our general view of distributed problem solving, we are encouraged to write domain specific strategies for agents as rules that are explicit, modular, hence modifiable.

ACKNOWLEDGMENTS

We are grateful to Lee Erman and Phil Klahr for comments on an earlier version of this paper.

REFERENCES

[1] Fikes, R., and Nilsson, N., STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2, (3), 189-208, 1971.

[2] Fahlman, S., A planning system for robot construction tasks. Artificial Intelligence, 5, (1), 1-49, 1974.

[3] Sacerdoti, E., A structure for plans and behavior. New York: Elsevier North-Holland, 1977.

[4] Erman, L, and Lesser, V., A multi-level organization for problem solving using many diverse cooperating sources of knowledge. Proceedings of the Fourth Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975, 483-490.

[5] Corkill, D., and Lesser, V., A goal-directed Hearsay-II architecture: Unifying data-directed and goal-directed control. COINS Tech. Rep. 81-15, University of Massachusetts.

[6] McArthur, D, Steeb, R., and Cammarata, S., A model of problem solving in domains with multiple active agents. In preparation, April, 1982.

[7] Goldberg, A., and Kay, A., The SMALLTALK-72 Instructional Manual, SSL-76-6, Xerox PARC, 1976.