Real-Time Problem Solving in The Phoenix Environment

Paul R. Cohen, Michael L. Greenberg
David M. Hart, Adele E. Howe

COINS Technical Report 90-28

Experimental Knowledge Systems Laboratory
Department of Computer and Information Science
University of Massachusetts, Amherst, MA 01003

ABSTRACT

Phoenix is a real-time, adaptive planner that manages forest fires in a simulated environment. To explore the issues of real-time problem solving, we have chosen a research methodology that emphasizes complex, dynamic environments and complete, autonomous agents. This paper describes the two components of the Phoenix system, a simulation testbed designed to facilitate the study of real-time problem-solving, and our own planning system that addresses real-time constraints with a variety of techniques.

Acknowledgments:
This research has been supported by DARPA/RADC F30602-85-C0014; ONR University Research Initiative Grant N00014-86-K-0764; AFOSR/RADC F30602-C-0008, subcontract 3539023; and a grant from the Digital Equipment Corporation. We also wish to thank Paul Silvey and Scott Anderson for their contributions to this work.
The authors choose alphabetical order as the one most representative of their equal contributions to the work reported here.

1.0 Introduction

Planning research in Artificial Intelligence is experiencing renaissance. In the past, AI planners generated plans but did not execute them; whereas now, we focus on both planning and execution, and we design methods to interleave them in a timely way. In the past, we assumed that planners could know the state of the world and the effects of all actions; whereas now, we recognize that the world is too big and noisy to apprehend completely and accurately, and although the effects of actions can be estimated, they are uncertain. In the past, we assumed that a planner was the only agent in an unchanging world; whereas now, we recognize that several agents may act simultaneously, competitively, or cooperatively, and the world itself changes according to its dynamics. In sum, we are developing planning methods for environments that are very much like our own physical world. The most salient characteristics of these environments, the ones that most urgently require us to rethink planning, are uncertainty and time. The Phoenix testbed provides an environment in which the success of planners depends on their ability to cope with uncertainty in a timely way.

The Phoenix project has built a testbed for real-time planning based on a simulation of forest fires and is building a planner for the fire environment. We will refer to these efforts as the Phoenix testbed and Phoenix planners, respectively. This paper briefly describes the testbed and the issues it raises for real-time planning, and then describes a planning system that addresses some of these issues.

2.0 The Forest Fire Environment and the Phoenix System

Phoenix's task environment is forest fire fighting in Yellowstone National Park. The Phoenix testbed simulates forest fires in the park. Figure 1 shows a monochrome screen dump of a portion of the fire map, which in this case is constructed from Defense Mapping Agency data for Yellowstone and the environs. (The fire map is usually displayed in color; the monochrome representation in Fig. 1 is missing a lot of detail, although one can see houses, roads, and rivers.) Yellowstone lake is in the lower part of the screen. In the first quadrant, one can see a fire partly surrounded by fireline cut by bulldozers. One technique for fighting forest fires is to bulldoze fireline that is wide enough that the fire can't cross it. Darker shading represents hotter fire. The Phoenix planner is directing the bulldozers (this planner-agent is the fireboss, whose location is marked by a **B** in the lower right). The circles surrounding them represent their field of view---everything beyond this circle is literally unknown to a bulldozer. In the upper center is a watchtower with a circular (and much wider) field of view. Although we can see the entire fire in Figure 1, the Phoenix planner sees only what its agents have seen, the fire within the field of view of the watchtower and all fire the bulldozers have seen up to now.

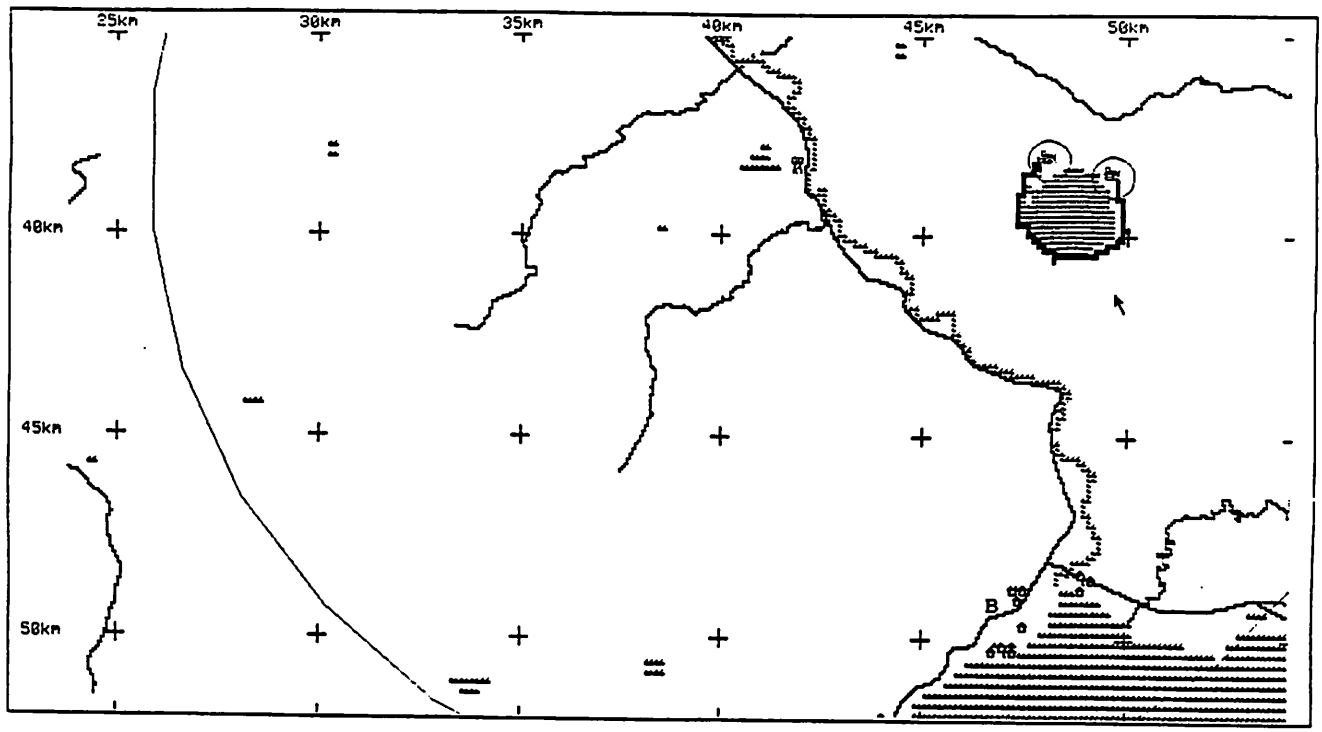

Figure 1: Two Bulldozers Fighting a Forest Fire in the Phoenix Simulation of Yellowstone National Park

2.1 Real Time in the Phoenix System

We can burn much of Yellowstone Park in about five minutes on a Texas Instruments Explorer II Lisp machine. This corresponds to several days of "real world" time. Phoenix can formulate a plan to contain a fire in a fraction of a second, whereas it would take a human fireboss many minutes or hours to do the same. Thus, in one sense, Phoenix is a real-time planner: it formulates plans fast enough for the physical world of forest fires. But this is not the sense of real time that interests us, because real-time planning is not an intellectual challenge unless time is short. We think of real-time planning as managing scarce temporal and physical resources. The Phoenix testbed must charge the Phoenix planner for thinking too long. The fire continues burning while the planner plans, and destroys not only real estate but, more importantly, the planner's confidence in the eventual success of its active plans. If it takes too long to plan for a small fire, the planner will end up confronting a large fire with an irrelevant plan.

To ensure that we will always be able to assess these costs, irrespective of the speed of planning algorithms and their supporting code, the Phoenix testbed allows us to control the ratio of planning time to simulated world time with a *real-time control knob*. This "knob" parameterizes the amount of cpu-time allotted to "cognitive" activities in relation to

elapsed simulation time - the real-world time of the simulation. For example, the knob can be set to allow the problem solving component of each agent one second of cpu-time (measuring actual cpu usage) for every minute of simulated time in the world. If our problem solver is successful at that level of temporal resources, we can increase the pressure by resetting the knob to two minutes of simulation time for every cpu-second, so that two minutes elapse in the world for every cpu-second the agent thinks. This effectively halves the time allowed for problem solving, testing the robustness of the planner to temporal constraints. Currently, one cpu second of planning time corresponds to five minutes of world time, which is stringent enough to stress the Phoenix planner. We can define real-time planning in terms of the real-time knob. We would not be satisfied if the performance of a real-time planner was extremely sensitive to the setting of the real-time knob. For example, if the world is "speeded up" to 10 minutes per cpu second, a Phoenix planner should still be able to cope. Real-time planning must be robust against small changes in the ratio of planning time to world time.

3.0 The Phoenix System Architecture

The Phoenix system has two parts: the Phoenix testbed, and the Phoenix planner. Our goal in designing the testbed was to specify a generic simulation environment and agent design that could be used by a number of researchers to develop different planning solutions in complex, real-time domains. We have created a discrete event simulator that can be used for any number of spatially and temporally distributed worlds. For example, the fire simulation could be replaced by an oil spill simulation, in which the map of Yellowstone would be replaced by a nautical chart of, say, Prince William Sound. We have attempted to design a generic agent architecture as part of the Phoenix testbed, but have not succeeded entirely in separating the generic characteristics from those specific to our planner. Thus, in the following sections we discuss the agent architecture as if the generic and specific characteristics are distinct, as was our goal, with the proviso that this is not yet the case.

The testbed consists of a discrete event simulator for environments with spatial and temporal extent, and a general agent architecture. Planning in Phoenix is distributed among the agents, and the "Phoenix planner" refers to the collective planning activity of all the agents. However, the Fireboss maintains a global view of the environment, forms global plans, and sends instructions to the other agents which plan according to the instructions and their local views. Section 3.1 describes the discrete event simulator. Section 3.2 describes both the general agent architecture (part of the testbed) and the Phoenix planner's specific implementation of agents using this shell. Section 3.3 describes the organization of fire-fighting agents in the Phoenix planner. A more complete description of the Phoenix system can be found in [1].

3.1 A Discrete Event Simulation of a Real-time World

The Phoenix testbed is built on top of a discrete event simulator that creates the illusion of a continuous world, where natural processes and agents are acting in parallel, on serial hardware. In the simulation, fire(s) burns continuously over time, affected by changing environmental conditions such as wind and precipitation. As the fire spreads, agents act in concert to control it. Some of these actions are physical, as in digging fireline and cutting trees. In parallel to these physical actions, agents are perceiving, moving, reacting to perceived stimuli, and thinking ahead about what action(s) to execute next. Maintaining this illusion of continuous, parallel activity on a serial machine is done by segregating each process and agent activity into a separate task, executing these tasks in small, discrete time units, and coordinating them by guaranteeing that no task ever gets too far ahead or behind the others.

3.2 Agent Architecture

There are four components to the architecture of an agent (see Figure 2). *Sensors* perceive the world. Each agent has a set of sensors for such perceptions as fire-location (are any cells within my radius-of-view on fire?). *Effectors* perform physical acts such as moving or digging fireline. *Reflexes* are simple stimulus-response actions, triggered when the agent is required to act faster than the time-scale of the cognitive component can handle. An example is the reflex by a bulldozer to stop if it is moving into the fire. The *cognitive* component performs mental tasks such as planning, monitoring actions, evaluating perceptions, and communicating with other agents. Each agent has these four components; each component can be endowed with a range of capabilities from limited (or none) to sophisticated.

Sensors get input from the world (fire simulation and map structures). Their output goes to state memory in the cognitive component, and also to the reflexive component (triggering instant responses in the form of short programs to the effectors). For example, a bulldozer sensor that detects fire within its radius-of-view updates state memory automatically. If the detected fire is in the path of the bulldozer, the emergency-stop reflex is also triggered. Effectors are programmed by the cognitive component and by reflexes. Their output performs actions in the world. In the preceding example, the emergency-stop reflex would program the movement-effector of the bulldozer to stop. If the fire were not too close, the cognitive component might then step in and program the same effector to move parallel to the fire. In addition, by programming the drop-blade effector to put the blade in the down position, the bulldozer would build fire-line as it moved. Sensors and effectors are first-class objects whose interactions with other components and the world are implemented in Lisp code. Reflexes, as mentioned, are triggered by sensory input, which causes them to program effectors to react to the triggering sensation. They are implemented in production-rule fashion, with triggering-sensations as their antecedent clauses and effector programs as their consequents. The cognitive component receives input from sensors and sends programs to the effectors to interact with the world. In addition,

communication is handled by passing messages among the cognitive components of the agents involved.

Sensors, effectors, and reflexes are synchronized closely with the simulation environment. This characteristic allows them to reflect most closely the current state of the world, but handicaps any efforts toward coordination and management of resources. The cognitive component is responsible for tasks related to coordination and management, the agent's reasoning. It operates in larger time slices, thus reducing the overhead of context switching, but increasing the possibility of reasoning from outdated information.

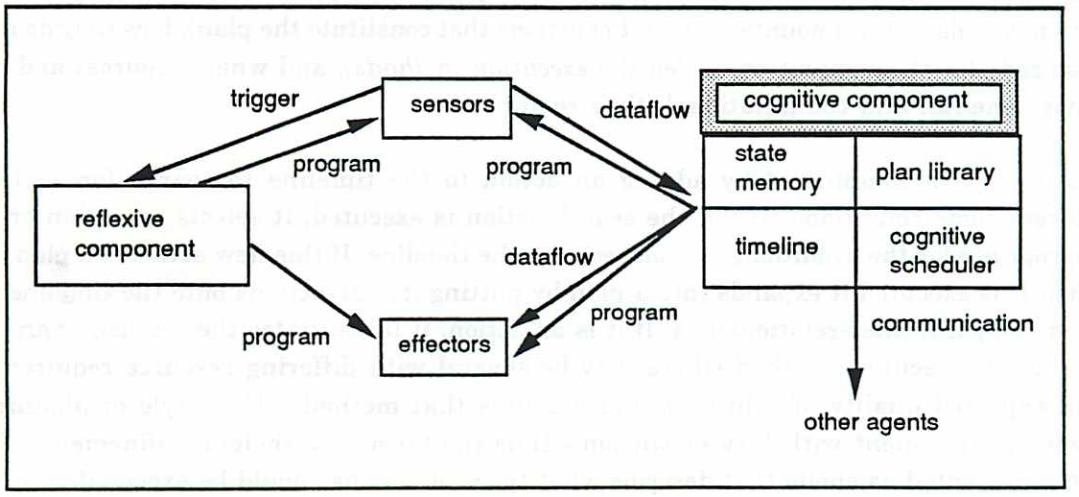

Figure 2: Phoenix Agent Architecture
(cognitive subcomponents are specific to the Phoenix planner)

The "generic" agent design of the Phoenix testbed expects little from the cognitive component other than a rudimentary ability to pick an action to execute, and arbitrate when more than one action is possible. For the Phoenix planner we have enhanced the cognitive component, in response to the real-time demands of the fire-fighting environment, to implement a style of planning we call *lazy skeletal refinement*. This planning style is common to all agents in the Phoenix planner, though it is flexible enough so that agents with a variety of cognitive capabilities are possible. For example, the different cognitive subcomponents (see Figure 2) can have any level of sophistication, and the relative cognitive sophistication among agents can be varied, so that a range of different agent designs are possible within this framework.

The Phoenix cognitive component directs its own actions by adding prospective actions onto the *timeline*, a structure for reasoning about the computational demands on the agent, then selecting and executing these actions one at a time.[1] Actions may be added in

[1]We are assuming that the cognitive component is a single serial processor because its implementation environment is. However, because the cognitive actions share resources, the problem

response to a change in environmental conditions (e.g., a new fire) or as part of the computation of other actions (e.g., through plan expansion). Every action that the cognitive component accomplishes is represented on the timeline with its temporal relations to other actions and possibly resource requirements. The *cognitive scheduler* decides which action to execute next from the timeline and how much time is available for its execution.

Actions may perform calculations, search for plans to address particular environmental conditions, expand plans into action sequences, assign variable values, process sensory information, initiate communication with other agents, or issue commands to sensors and effectors. These actions are represented in skeletal form in the *plan library*. Actions are described by what environmental conditions they are appropriate for, what they do (if the action is a plan, this amounts to the set of actions that constitute the plan), how they do it (the Lisp code for their execution, called the *execution methods*), and what resources and data, environmental and computational, they require.

Planning is accomplished by adding an action to the timeline to search for a plan to address some conditions. When the search action is executed, it selects an action or plan appropriate for the conditions and places it on the timeline. If this new action is a plan, then when it is executed it expands into a plan by putting its sub-actions onto the timeline with their temporal inter-relationships. If it is an action, it instantiates the requisite variables, selects an execution method (there may be several with differing resource requirements and expected quality of solution), and executes that method. This style of planning is skeletal refinement with lazy expansion - thus the term lazy skeletal refinement. Plans are represented as shells that describe what types of actions should be executed to achieve the plan but do not include the exact action or its variable values until it is executed. Delaying expansion allows the expanded plan to address more closely the actual state of the environment during execution.

This style of planning and acting is designed to be responsive to a complex dynamic world by postponing decisions on exactly what action to take, while also grounding potential actions in a framework (a skeletal plan) that accounts for data, temporal and resource interactions, and by operating at both a reflexive and cognitive level. The combination of a reflexive and cognitive component accounts for time scale mismatches inherent in an environment that requires micro actions, like following a road, involving quick reflexes and little data integration and contemplative processing, like route planning, involving long search times and integration of disparate data such as available roads, terrain conditions, and fire reports. We have designed an agent architecture that in effect combines two different planning components: one highly reactive, triggered by specific environmental stimuli and operating at very small time scale, and the other slower and more contemplative, integrating large amounts of data and concerned with resource management and coordination.

---

prevents complete parallel execution by requiring some arbitration between actions that make similar, but conflicting changes to shared resources.

3.3 Organization of Phoenix Fire-fighting Agents

The Phoenix testbed has been designed to support different kinds of planners. The architecture is generalized to allow experimentation with various capabilities and organizations of planning agents, as in the distributed planner being built by Victor Lesser's group at UMass. Our own Phoenix planner is centralized; one fireboss agent directs all the other agents' activities, sending them directives and receiving status reports. From the status reports, which include fire sightings, position updates, actions completed, etc., the fireboss pieces together a global view of the fire situation. It chooses a plan from its plan library based on this situation assessment and communicates action specifications (steps in the plan) to the various agents at its disposal. These specifications are used by the agents as keys to look up plans in their plan libraries. Thus, the planning method is the same -- indexed plan look-up; the difference between the fireboss and the field agents is the view of the world and the plans each knows. Each of the other agents is limited in its view to what it has perceived, and only has plans for actions it can perform. There is currently no communication among the field agents -- all communication is between the fireboss and individual agents.

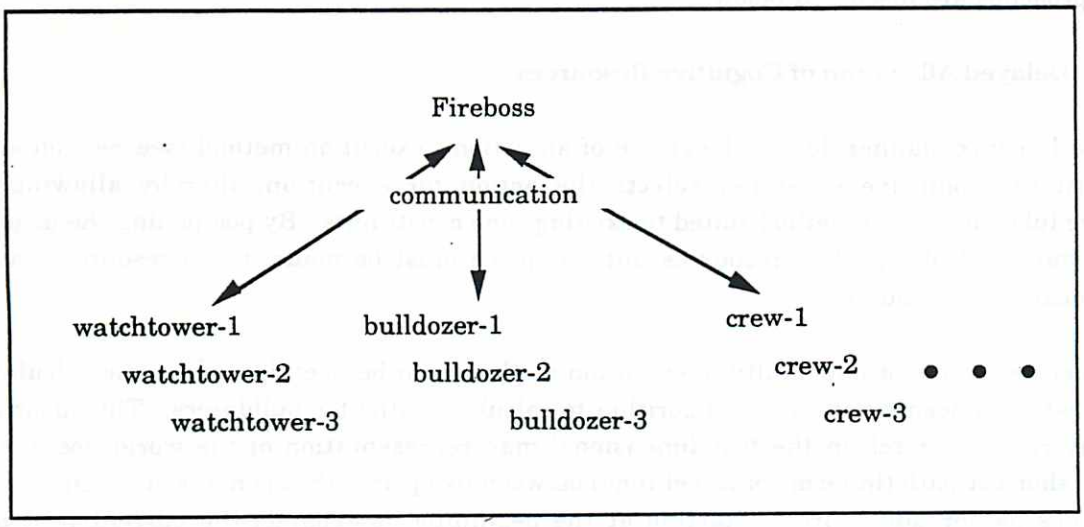

Figure 3: Centralized Organization of Fire-fighting Agents in Phoenix Planner

Clear lines of authority and division of responsibilities exist among agents in the Phoenix planner. The fireboss maintains the global view, based on the local views of its agents, and "acts" in the world by sending directives to the agents. It executes cognitive plans whose effects are to gather information, send directives to agents, and coordinate their activity via communications. In contrast, the agents execute cognitive plans whose actions program sensors and effectors, which in turn effect physical actions in the world. In some sense the fireboss is a "meta-agent" responsible for maintaining global coherence; its sensors and effectors are other agents, who gather its percepts and effect its intended actions.

4.0 Real-time Control in Phoenix

A number of design decisions in the Phoenix testbed have been made specifically to facilitate real-time control. One important decision is to incorporate both reflexive and cognitive abilities in agents, enabling agents to respond reflexively to events that occur quickly, while responding more deliberately to resource management and coordination problems on a longer time scale. Another is the real-time control knob, allowing us to force real-time pressure on cognitive activities. How does the Phoenix planner respond to real-time pressure? The planner uses several approaches to managing cognitive resources. One is to provide multiple execution methods for timeline entries, so that there is always a range of choices that vary in their timeliness (among other features). This is an integral part of the lazy skeletal refinement planning method. Another approach is through control of the cognitive scheduler. We are working on different scheduling strategies for managing the actions on the timeline that will provide greater responsiveness to real-time constraints. A third approach is an expectation-based monitoring technique whose goals are to reduce the overhead of monitoring while providing early warning of plan failure, allowing the planner more time to adjust. These approaches are discussed below.

4.1 Delayed Allocation of Cognitive Resources

The Phoenix planner delays the choice of an action's execution-method (see Section 4.2.2) until the cognitive scheduler selects the action for execution, thereby allowing the scheduler to select a method suited to existing time constraints. By postponing the ultimate commitment of cognitive resources until a choice must be made, those resources can be allocated judiciously.

A good example of how multiple execution-methods can be useful involves the calculation of paths. Phoenix uses an A* algorithm to calculate paths for bulldozers. This algorithm performs A* search in the two-dimensional map representation of the world, looking for the shortest path (in terms of travel time) between two points that is not obstructed (by such things as spreading fire). Starting at the beginning, it expands the current best path incrementally, searching each unobstructed neighboring cell for the best next step. It is parameterized to work at multiple levels of resolution, so that search steps could range from 128 meters up to 8 kilometers. A small resolution yields the shortest (quickest) path, and is therefore the least time consuming for the bulldozer. However, this resolution requires the most computation (i.e., cognitive resources). The highest resolution typically yields a longer (more time consuming) path, but this path can be calculated quickly. At times it even fails to find a solution, since there are bottlenecks in the map that it can't find. Each of these resolutions constitutes a different execution-method for calculating a path. They span the range of the trade-off between cognitive-time and quality of solution: the least expensive method from the standpoint of cognitive resources yields the worst solution, and the most expensive method yields the best. A balanced solution lies somewhere in between.

4.2 Scheduling Algorithms (Cognitive Scheduler)

The cognitive scheduler is responsible for selecting the next action from the timeline that should be executed, choosing an execution method for the action, and executing it (see Section 3.2). Thus, the scheduler is key to controlling the responsiveness of the cognitive component to real-time constraints. When time is short, the scheduler should choose wisely among the actions ready for execution, and should pick execution methods for those actions that emphasize speed over other features. There are a number of considerations for the scheduler besides the choice of execution method. One is the choice of scheduling algorithm. There are many algorithms with known characteristics that we can exploit. The scheduler will be designed to switch strategies (algorithms) under different conditions, exploiting the strengths of each according to the situation. Switching algorithms will involve some overhead in adjusting data structures, but hopefully this will be outweighed by the benefits of having a strategy suitable to the constraints of the moment. Another consideration is the severity of deadlines in the system. Since the environment is ongoing, there are many kinds of deadlines. If two bulldozers fail to rendezvous at a point before the fire reaches it, they can often pick a slightly removed point and still achieve their goal. On the other hand, a bulldozer can be trapped between a spreading fire and a body of water and have no escape unless the fireboss provides a safe exit route (based on the global view of the fire). Thus, some knowledge about the relative types and importance of deadlines can help in the scheduling process, providing precedence criteria when all scheduling constraints can't be met.

4.3 Envelopes - Early Prediction of Failure and Reducing Monitoring Time

Monitoring follows plan progress and changes in the world. Many plans have deadlines and coordination points for actions. The coordinating agent must keep track of executing actions and be sure they are progressing as intended. Monitoring of plan execution involves several things. Is the plan unfolding in a satisfactory way? More specifically, is the progress of each plan step satisfactory with respect to the constraints of the plan? How is the world changing? Are the changes within the range of what was anticipated, or must new events, or differences in rate of change, be considered for their effects on the plan? Monitoring these situations consumes costly cognitive resources. The monitoring agent must follow each plan step and watch the world constantly, assessing ramifications for the plan of rates of progress or changes in the world, and trying to decide when the current plan (or a portion) will or has failed.

To make the Phoenix planner responsive in real-time, we've structured monitoring to address both problems by developing an expectation based monitoring structure called an *envelope*. An envelope is a functional representation of a dynamic process (such as a plan step or environmental conditions). It relates expectations to predicted rates of change. Expectations are created from constraints and projection. If an agent must be at a certain place at a certain time, we can tell by projection whether the deadline is feasible - can the agent travel the distance in the given time? By projecting the expected time of travel (based on a parameter such as average speed for the agent on the given terrain), we can create an

envelope for the travel time and use it to monitor the progress of the agent in meeting the deadline. The envelope *tracks the rate-of-progress* of the agent (has it traveled as far as was expected in the elapsed travel time?). It also *predicts the expected arrival time*, based on the recent progress of the agent. Furthermore, it *predicts the minimum speed at which the agent must travel over the remaining distance* to arrive before the deadline. If this speed is at or approaching the top speed of the agent, then the envelope signals the planner that the deadline is in jeapordy. Thus, we have an early warning of failure. By setting a threshold on the highest average speed that is likely, the envelope can warn the planner that the plan step may fail when the minimum required speed exceeds it. When the minimum required speed exceeds the maximum possible speed of the agent, the envelope signals plan failure.

Envelopes can also be used with processes in the world whose rates of change affect the success of plans. One of the considerations for choosing a plan is the expected rate of spread of the fire. An expected rate of spread is established by projection, and is based on the interplay of contributing factors such as ground cover and wind speed. A plan is chosen that can control the fire based on this expectation. In addition, an envelope is created for the expectation. Thus, if one of the factors affecting fire spread changes dramatically, this envelope will predict that the *expected* rate of spread of the fire is wrong, which signals to the planner that it should form a new expectation about the fire's rate of spread and reassess whether the chosen plan is still adequate.

As long as predicted progress remains within expectations, the planner leaves monitoring to envelopes, which take in new facts, calculate progress and compare progress to expectations. Thus the planner doesn't spend its time assessing minor changes in the situation for their impact on the current plan - it only responds when an envelope is violated and issues a warning. Furthermore, the predictive component of envelopes is an early warning system, since threshold warning levels can be set to indicate when an action is in danger of failing. This warning allows the planner to start contingency planning as soon as possible, before plan failure actually occurs.

5.0 Conclusion

The Phoenix testbed simulates a complex, real-time environment (forest-fires in Yellowstone National Park) and provides an architecture for building planning agents to fight the fires. The Phoenix planner is an instance of this agent architecture, combined with an organization of various types of agents (all with the same basic design) developed to study the interaction of agents with this environment. A number of design decisions have been made to address the real-time management of agents' resources, including a combination of reflexive and cognitive planning capabilities tailored to differences in time-scales inherent in this environment, a style of planning called lazy skeletal expansion which delays the allocation of cognitive resources until the environmental constraints on those resources are known, a flexible scheduler that employs various scheduling strategies depending on environmental constraints, and a monitoring technique called envelopes that minimizes the cognitive resources devoted to monitoring

plan execution while providing early warning of failure. In addition, the simulation includes a real-time control knob that allows the user to adjust the temporal constraints imposed on the cognitive component of the agent architecture, thereby testing the robustness of planning techniques to real-time problem-solving.

References

1. Cohen, P. R., Greenberg, M. L., Hart, D. M. and Howe, A. E., "Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments", to appear in *AI Magazine*, Fall, 1989, and also EKSL Technical Report 89-61, Department of Computer and Information Science, University of Massachusetts, Amherst. 1989.