Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments[1]

Paul R. Cohen, Michael L. Greenberg, David M. Hart, Adele E. Howe[2]

COINS Technical Report 89-61

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

[1]This research has been supported by DARPA/RADC # F30602-85-C0014; the Office of Naval Research, under a University Research Intitiative grant, ONR N00014-86-K-0764; the Office of Naval Research, contract # N00014-87-K-0238; and a grant from the Digital Equipment Corporation. We also wish to thank Paul Silvey.

[2]The authors choose alphabetical order as the one most representative of their equal contributions to the work reported here.

Abstract

Phoenix is a real-time, adaptive planner that manages forest fires in a simulated environment. Alternatively, Phoenix is a search for functional relationships between the designs of agents, their behaviors, and the environments in which they work. In fact, both characterizations are appropriate, and together exemplify a research methodology that emphasizes complex, dynamic environments and complete, autonomous agents. Within the Phoenix system we empirically explore the constraints the environment places on the design of intelligent agents. This paper describes the underlying methodology and illustrates the architecture and behavior of Phoenix agents.

1. The Phoenix Research Agenda

The Phoenix project is directed by three complementary goals. First, there are immediate technical aims: a real-time, adaptive planner for controlling simulated forest fires, approximate scheduling algorithms for coordinating multiple planning activities, knowledge representations for plans and for measuring progress toward goals, and distributed planning algorithms. Secondly, there are motivating issues, of which the foremost is to understand how complex environments constrain the design of intelligent agents. We seek general rules that justify and explain why an agent should be designed one way rather than another. The terms in these rules describe characteristics of environments, tasks and behaviors, and the architectures of agents. Lastly, because AI is still inventing itself, Phoenix is a commentary on the aims and methods of the field. Our position is that most AI systems have been built for trivial environments that offer no constraints on their design, and thus no opportunities to learn how environments constrain and inform system design [4] . To afford ourselves this opportunity, we began the Phoenix project by designing a real-time, spatially-distributed, multi-agent, dynamic, ongoing, unpredictable environment.

In the following pages we will describe Phoenix from the perspective of our technical aims and our motives. Section 2 describes the Phoenix task---controlling simulated forest fires---and explains why we use a simulated environment instead of a real, physical one. Section 3 discusses the characteristics of the forest fire environment and the constraints they place on the design of agents. The two lowest layers of Phoenix, described in Section 4, implement the simulated environment and maintain the illusion that the forest fire and agents are acting simultaneously. Above these are two other layers: a specific agent design (Sec. 5), and our organization of multiple fire-fighting agents (Sec. 6). These sections describe how Phoenix agents plan in real time, but do not provide the minute detail that is offered elsewhere [3]. Section 7 is an example of Phoenix agents controlling a forest fire. Section 8 describes the current status of the project and our immediate goals.

2. The Problem

The Phoenix task is to control simulated forest fires by deploying simulated bulldozers, crews, airplanes, and other objects. We will discuss how the simulation works in Section 4, concentrating here on how it appears to the viewer and the problems it poses planners.

The Phoenix environment simulates fires in Yellowstone National Park, for which we have constructed a representation from Defense Mapping Agency data. Figure 1 shows a view of an area of the park; the grey region at the bottom of the screen is the northern tip of Yellowstone Lake. The thick grey line that ends in the lake is the Yellowstone River. The Grand Loop Road follows the river to the lake, where it splits. The large "B" in the bottom left corner marks the location of the fireboss, the agent that directs all others. Two bulldozers are shown building fireline around a fire in this figure.[1]

Fires spread in irregular shapes, at variable rates, determined by ground cover, elevation, moisture content, wind speed and direction, and natural boundaries. For example, fires spread more quickly in brush than in mature forest, are pushed in the direction of the wind and uphill, burn dry fuel more readily, and so on. These conditions also determine the probability that the fire will jump fireline and natural boundaries. But for two exceptional conditions (convective and crown fires), Phoenix is an accurate simulator of forest fires. Fire-fighting objects are also simulated accurately; for example, bulldozers move at a maximum speed of 40 kph in transit, 5 kph travelling cross-country, and 0.5 kph when cutting fireline. To give a sense of scale, the fire in Figure 1 is about 1.5 kilometers in diameter and has burned for about eight simulated hours. The fire's history, which can be read in Figures 2, 3, and 4 is as follows: At noon in simulation time (Fig. 2) a fire was ignited, and later detected by a watchtower (not visible in the figures). A little later, two bulldozers started a journey from the firestation, marked by a "B" in the southwest corner, to the rear of the fire. Because the wind was from the southeast, the rear was southeast of the fire. At 3 p.m. (Fig. 3) the bulldozers arrived at the rear and started cutting fireline. Figure 4 was generated at 8 p.m., simulation time. The fire was contained a little later. This entire simulation took about 1 minute on a TI Explorer.

Fires are fought by removing one or more of the things that keep them burning: fuel, heat, and air. Cutting fireline removes fuel. Dropping water and flame retardant removes heat and air, respectively. In major forest fires, controlled backfires are set to burn areas in the path of wildfires and thus deny them fuel. Huge "project" fires, like those in Yellowstone last summer, are managed by many geographically dispersed firebosses and hundreds of firefighters.

The current Phoenix planner is a bit more modest. One fireboss directs a few bulldozers to cut line near the fire boundary. We currently lack but are implementing "indirect" attacks, which exploit natural boundaries as firebreaks (such as the river in Fig. 1) and "parallel" attacks, which involve backfires. The Phoenix planner does use common fire-fighting plans, such as the *two bulldozer surround* illustrated in Figures 2, 3, and 4. In

[1]Much available information is not displayed in the monochrome interface to Phoenix. The color interface displays ground cover and elevation contours, giving the user a better picture of the terrain over which the fire is spreading.

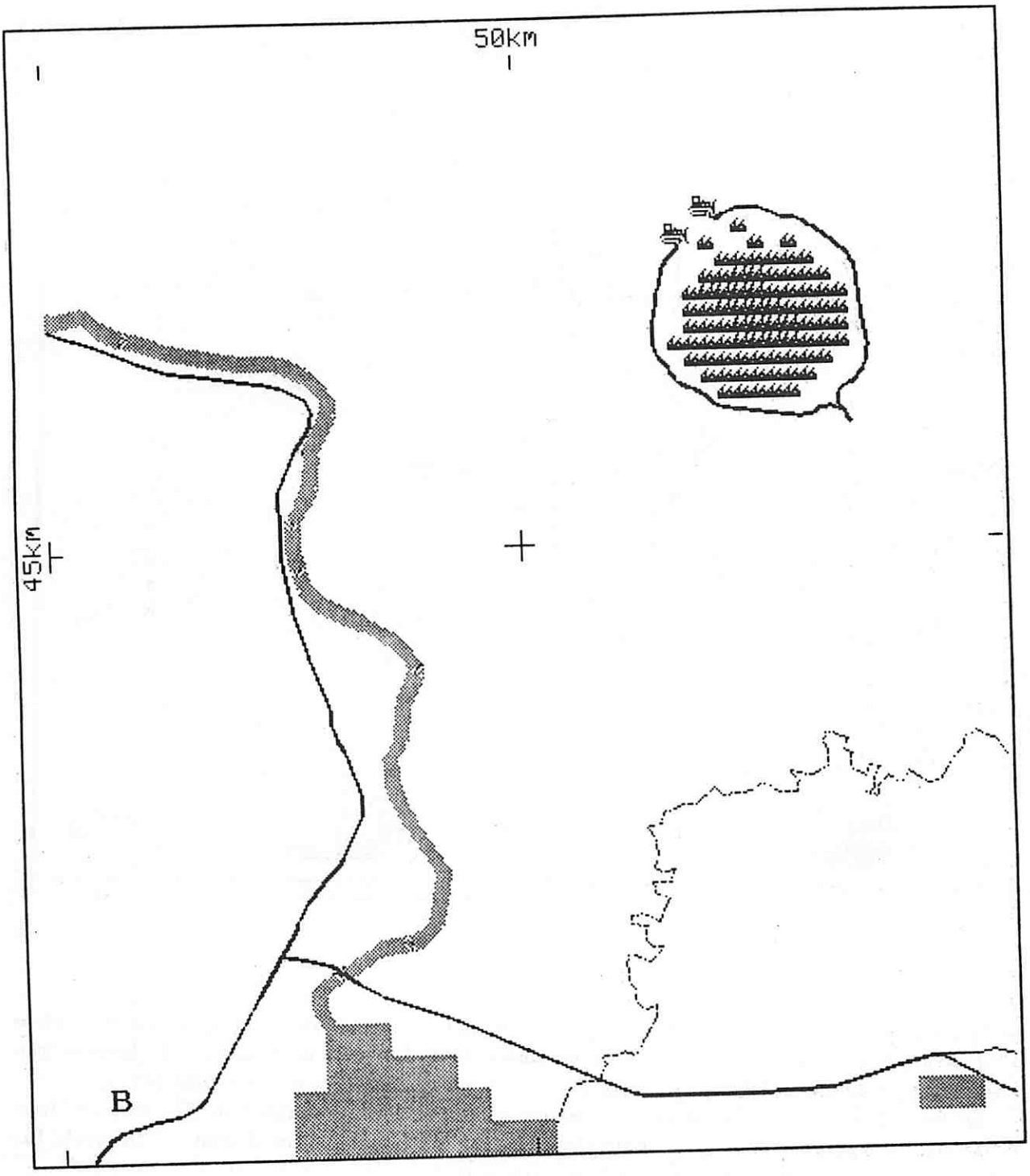

Figure 1: A portion of Yellowstone National Park as viewed in the Phoenix simulator

This is a small fraction (about 7 km. by 8 km.) of the entire 75 km. square map. The northern tip of Yellowstone Lake appears at the bottom (gray shading represents water). The Yellowstone River empties into the lake here, as does a smaller stream called Pelican Creek (meandering line in lower right). Grand Loop Road runs along the lake and river from south to north. East Entrance Road cuts across above the lake from west to east. The large B *marks the fireboss and bulldozer base. In this frame two bulldozers have almost surrounded a fire with fireline. The fire is burning at different intensities; the inner part has been burning longer and is hotter (note the darker icon). The kilometer markings in the margins show distances east and south from the northwest corner of the park.*

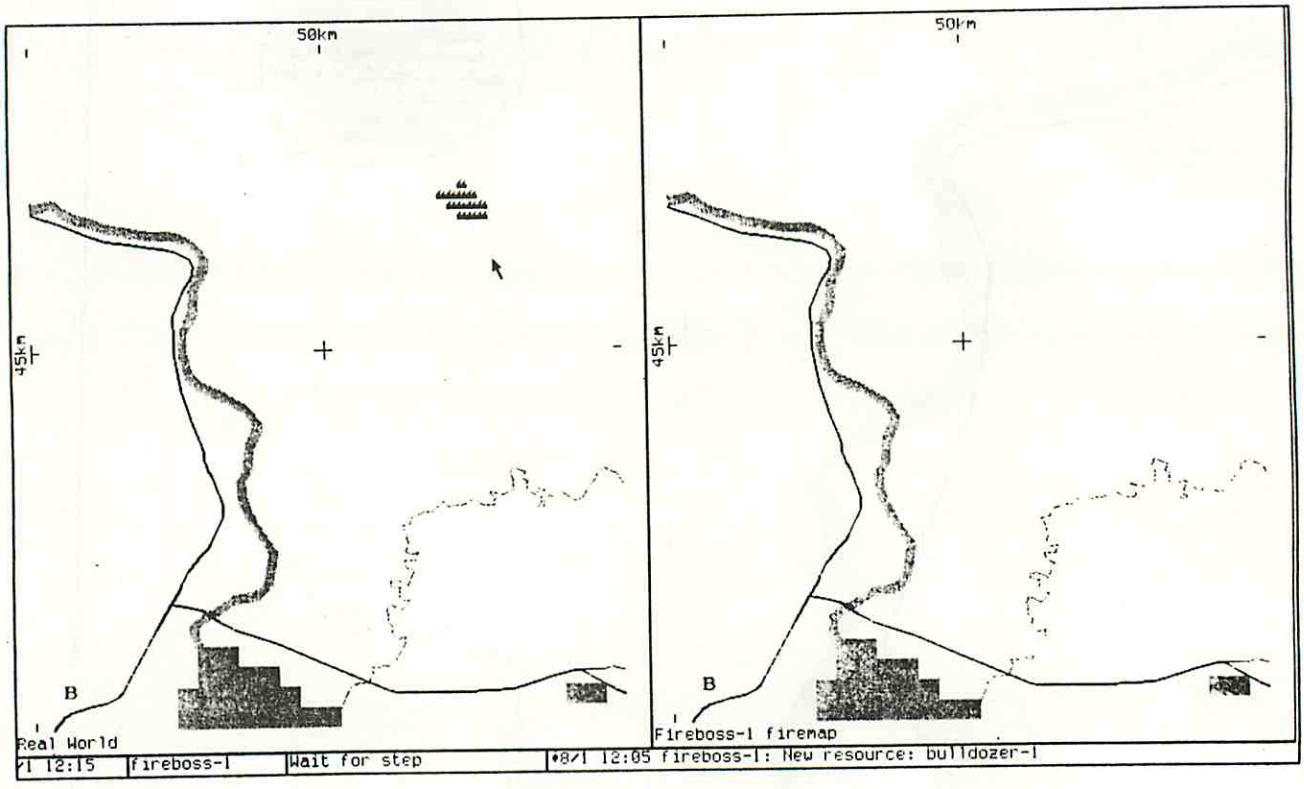

Figure 2: Fire at 12:15 pm

The left pane displays the real world; the right pane displays the current state of the world as the fireboss "sees" it. A fire has started and is displayed in the upper right of the real world pane. The fireboss, who finds out about fires from watchtower reports, doesn't know about this fire yet (see right pane).
The status bar below the two panes shows information about the running simulation. The date and time are in the left box (partly obscured). All simulations start at 12:00 noon on August 1. The right box displays timestamped information messages from various tasks.

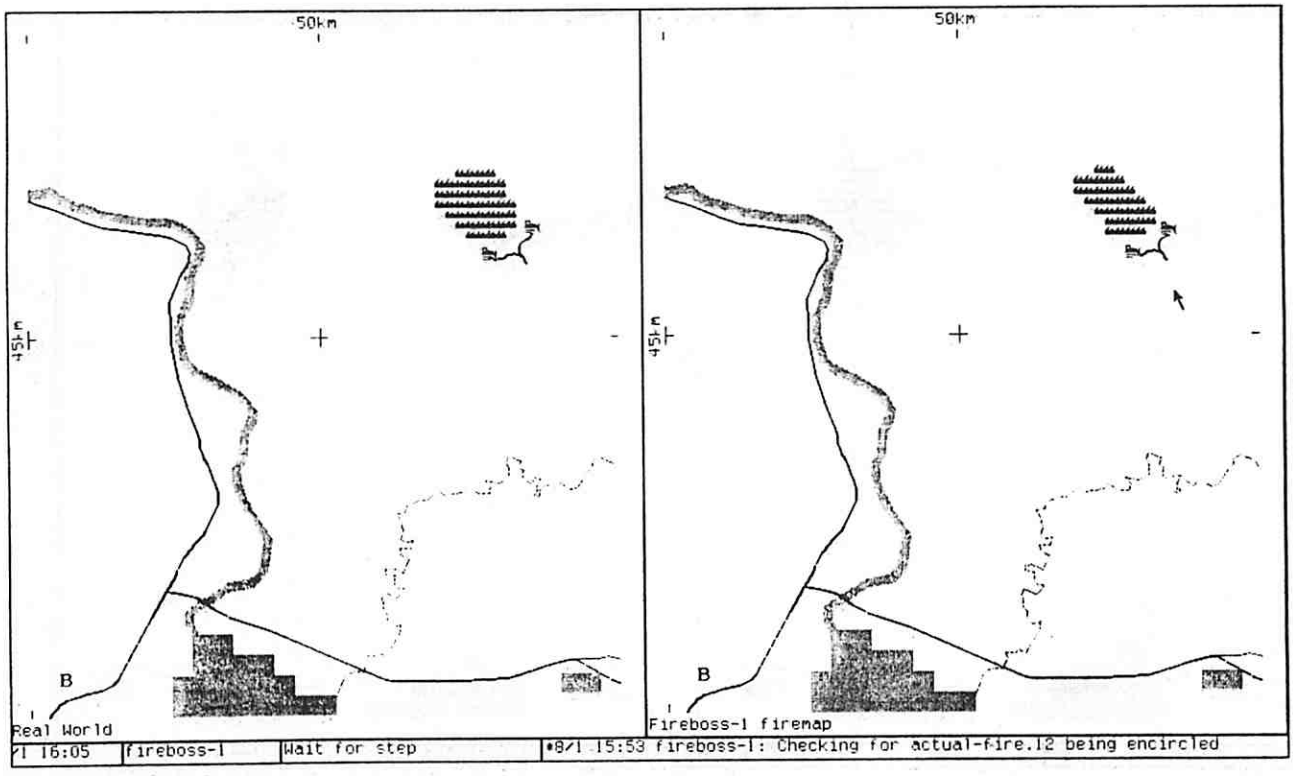

Figure 3: Fire at 4:05 pm

After four hours, two bulldozers have reached the fire and are beginning to build fireline around it. The two bulldozer plan was chosen by the fireboss based on environmental factors such as the size of the fire and the wind characteristics. Note that the fireboss's view of the situation is still slightly outdated. It sees fewer of the burning cells and isn't aware of all fireline that has been dug. It learns about these events from status reports sent by agents.

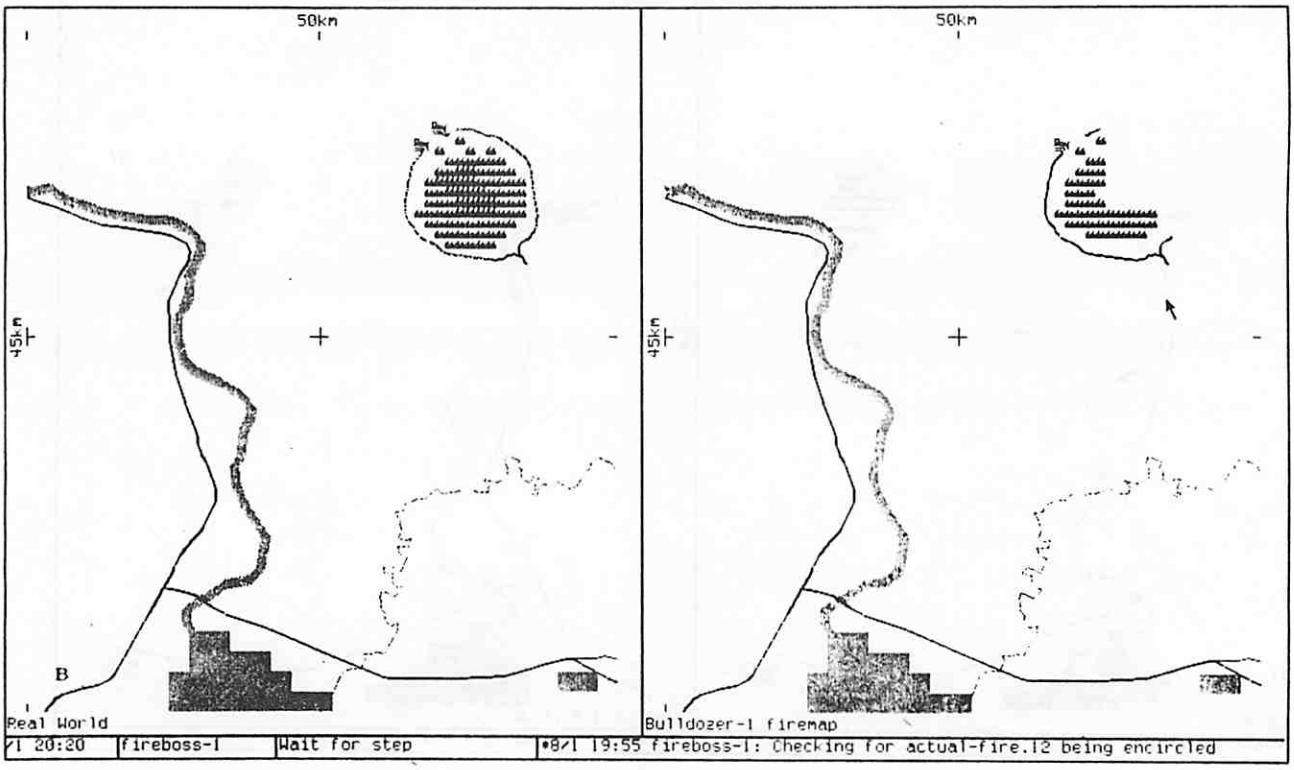

Figure 4: Fire at 8:20 pm

After 8 hours the fire is nearly encircled. The bulldozers are close to meeting at the fire front. The left pane again displays the real world; the right pane displays Bulldozer-1's view. It knows about the part of the fire it passed while digging line, as well as the fireline dug within its field of view (some of which was dug by the other bulldozer).

this plan, two bulldozers begin at the rear of the fire and work their way around to the front, pinching it off.

The Phoenix fireboss directs bulldozers but does not control them completely. In fact, the fireboss gives fairly crude directions, such as "go to location x,y," and individual agents decide how to interpret and implement them. Thus, bulldozers and other agents are semi-autonomous. Other organizational structures are enabled by increasing or decreasing the degree of autonomy; for example, an earlier fire planner, designed by David Day, had a single fireboss that controlled every action of all its agents. At the other extreme, we are working with Victor Lesser on a completely distributed version of Phoenix, in which agents negotiate plans in the absence of a single fireboss. We can experiment with different organizational structures because all agents have exactly the same architecture, and so each can assume an autonomous, semi-autonomous, or completely subservient role.

Although Phoenix agents and their environment are all parts of a large software system, we have designed them to give the impression of independent agents "playing against" simulated forest fires, much as we would play a video game. In fact, early in the project, we built an interface to allow us, instead of an automated planner, to direct fire fighting agents. It required us to control several agents simultaneously, and demanded considerable foresight and planning. We found it impossible to control more than a couple of bulldozers in real time in the vicinity of the fire, so we gave bulldozers simple reflexes, enabling them to scurry away from encroaching fire. Since then, the basic style of interaction between the Phoenix environment and the Phoenix planners has not changed: One or more planners, AI or human, direct semi-autonomous agents to move around a map, building line around continuously burning fires.

The decision to develop and test Phoenix agents in a simulated environment is, to some, profoundly wrong. One argument is that by building the environment and the interface to agents, we risk deferring or ignoring difficult problems. For example, if we build a simulated agent that has a completely accurate internal map of its simulated environment and, when it moves, its "wheels" don't slip, then all its planning and acting can be dead-reckoning. Of course we *can* create trivial environments and develop techniques that won't work in real environments, but why would we? The point of using simulators is to create more realistic and challenging worlds, not to avoid these challenges. In response to the criticism that simulators can never provide faithful models of the real, physical world, we argue that the fire environment is a real-time, spatially-distributed, ongoing, multi-actor, dynamic unpredictable world - irrespective of whether it is an accurate model of how forest fires spread. As it happens, the fire environment is an accurate model of forest fires, but this isn't necessary for the environment to challenge our current planning technology. Moreover, we want to leave open the possibility of working in simulated worlds that are unlike any physical world that we have encountered.

The advantages of simulated environments are that they can be instrumented and controlled, and provide variety; all essential characteristics for experimental research. Specifically, simulators offer these advantages:

Control. Simulators are highly parameterized, so we can experiment with many environments. For example, we can change the rate at which wind direction shifts, or speed up the rate at which fire burns, to test the robustness of real-time planning mechanisms. Most important, from the standpoint of our work on real-time planning, is the fact that we can manipulate the amount of time an agent is allowed to think, relative to the rate at which the environment changes, thus exerting (or decreasing) the time pressure on the agent (Sec. 4).

Repeatability. We can guarantee identical initial conditions from one "run" to the next; we can "play back" some histories of environmental conditions exactly, while selectively changing others.

Replication. Simulators are portable, and so enable replications and extensions of experiments at different laboratories. They enable direct comparisons of results, which would otherwise depend on uncertain parallels between the environments in which the results were collected.

Variety. Simulators allow us to create environments that don't occur naturally, or that aren't accessible or observable.

Interfaces. We can construct interfaces to the simulator that allow us to defer questions we'd have to address if our agents interacted with the physical world., such as the vision problem. We can also construct interfaces to show things that aren't easily observed in the physical world; for example, we can show the different views that agents have of the fire, their radius of view, their destinations, the paths they are trying to follow, and so on. The Phoenix environment graphics make it easy to see what agents are doing and why.

3. Environmental Constraints on Agent Design

From the preceding descriptions of the Phoenix environment and tasks, one can begin to see the challenges they present to Phoenix agents. Our challenge, as researchers, is to design these agents for the Phoenix environment. The relationships between agent design, desired agent behaviors, and environment characteristics are clarified by what we call the behavioral ecology triangle, shown in Figure 5.

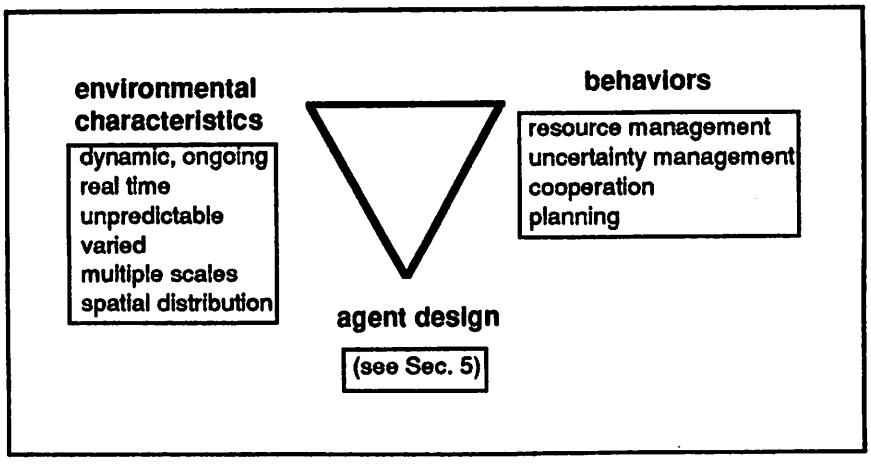

Figure 5: The behavioral ecology triangle.

The vertices of the triangle are the agent's design (i.e., internal structures and processes), its environment, and its behavior (i.e., the problems it solves and the ways it solves them). In this context, our tasks (and, indeed, the tasks of all AI research on intelligent agents) are:

Environment Analysis: What characteristics of an environment most significantly constrain agent design?

Design: What architecture will produce the desired behaviors under the expected range of environmental conditions?

Prediction: How will a particular agent behave in particular environmental conditions?

Explanation: Why does an agent behave as it does in particular environmental conditions?

Generalization: Over what range of environmental conditions can we expect particular behaviors from the agent? Over what range of problems? Over what range of designs?

To date, the Phoenix project has concentrated on environment analysis (see below), the design task (Secs. 5 and 6) and on building an environment in which the other tasks can be empirically pursued. Figure 5 implicitly captures many hypotheses and explanatory tasks. We can think of "anchoring" two corners and "solving for" a third; for example, we can anchor an environment and a set of behaviors and solve for an agent design. Or we can anchor a design and an environment and test predictions about behavior. Another more exploratory research strategy is to anchor just one corner, such as the environment, and look for tradeoffs in the other corners. For example, given the Phoenix environment, how is adaptability to changing time pressures affected by the design decision to search for plans in memory, rather than generate them from scratch?

Let us survey the characteristics of the Phoenix environment that constrain the design of Phoenix agents, and the behaviors the agents must display to succeed at their tasks. The fire environment is *dynamic* because everything changes: wind speed and direction, humidity, fuel type, the size and intensity of the fire, the availability and position of fire-fighting objects, the quantity and quality of information about the fire, and so on. The environment is *ongoing* in the sense that there isn't a single, well-defined problem to be solved, after which the system quits, but rather, there is a continuous flow of problems, most of which were unanticipated. The environment is *real-time* in the sense that the fire "sets the pace" to which the agent must adapt. The agent's actions, including thinking, take time, and during that time, the environment is changing. These characteristics require an agent to have some concept of relative or passing time. The agent must reason about the potential effects of its actions, and particularly about how much time those actions may require. Additionally, it must be able to perceive changes in its environment, either directly through its own senses or indirectly through communication with other agents.

The environment is *unpredictable* because fires may erupt at any time and any place, because weather conditions can change abruptly, and because agents may encounter

unexpected terrain, or fire, or other agents as they carry out plans. An agent must respond to unexpected outcomes of its own actions (including the actions taking more or less time than expected) and to changes in the state of the world. This requires interleaving planning, execution and monitoring and suggests that detailed plans of long duration will be likely to fail before successful completion. The unpredictability of the environment requires agents to be flexible, particularly in the way they handle temporal resources. In fact, all resources, including time, fire fighting agents, money, fuel, and equipment, are limited and non-renewable. Because the environment is ongoing, decisions about resources have long-term effects that constrain later actions, and require agents to manage their resources intelligently, with a global perspective. For this reason, among others, Phoenix agents cannot be exclusively "reactive."

Whereas unpredictability is a characteristic of the Phoenix environment, *uncertainty* arises in agents. Uncertainty is partly due to the fire continuously moving, partly because changes in wind speed and direction are unpredictable, partly due to communication delays between agents, and partly because individual agents have very limited views of the world. For example, to the northeast of Bulldozer 1, in the right-hand pane of Figure 4, there is a small black patch of fireline. This is all Bulldozer 1 knows about the location and progress of the other bulldozer (whose actual location is shown in the left-hand pane of Fig. 4), and illustrates how far Bulldozer 1 can see. It follows that Bulldozer 1's firemap, as shown in the right-hand pane, must merge what it currently sees with what it recalls. As one would expect, the recollection is inaccurate; Bulldozer 1 thinks the fire at its southern point is a few hundred meters from the fireline, because that's where it was when Bulldozer 1 cut the fireline. In fact, the fire has spread all the way to the fireline, as shown in the left-hand pane. As a consequence of these types of uncertainty, agents must allot resources for information gathering. Agents must be able to integrate and disseminate local information, and, because of their own localized views, they must be able to communicate and coordinate with each other.

The fact that events happen at *different scales* in the Phoenix environment has profound consequences for agent design. Temporal scales range from seconds to days, spatial scales from meters to kilometers. Agents' planning activities also take place at disparate scales; for example, a bulldozer agent must react quickly enough to follow a road without straying due to momentary inattention, and must also plan several hours of fire-fighting activity, and must do both within the time constraints imposed by the environment.

Given the size and variation in the world map, the degree to which the environment can change, and the possible actions of agents, the environment can produce a large *variety* of states. Consequently, an agent must know how to act in many different situations. The ramifications for agent design depend on whether small differences in environmental conditions can produce large differences in the utilities of plans. For example, if every fire scenario is truly different in the sense that each requires a unique, scenario-specific plan, then it may be pointless to provide agents with memories of previous plans. In fact, we believe that although the fire environment presents a wide variety of states, these differences do not require radically different plans.

The Phoenix environment is *spatially distributed*, and individual agents have only limited, local knowledge of the environment. Moreover, most fires are too big for a single agent to control; their perimeters grow much faster than a single agent can cut fireline.

These constraints dictate multi-agent, distributed solutions to planning problems. They also expand the scope of our research from a study of *agent* design to a study of *organizational* design. We have drawn a line, temporarily, and excluded the latter.

In sum, to perform their designated tasks, under the constraints of the Phoenix environment, Phoenix agents must engage in particular behaviors. In gross terms, these are resource management, uncertainty management, planning, and cooperative problem-solving; more specific behaviors have just been discussed. The question we address in Sections 5 and 6 is how do the characteristics of the Phoenix environment, in concert with the desired behaviors of Phoenix agents, constrain the design of the agents? Specifically, what architecture is capable of planning in real time, responding to events at different time scales, coordinating the efforts of several agents, collecting and integrating data about a changing environment, and so on.

4. The Phoenix Environment, Layers 1 and 2.

To facilitate experiments, Phoenix is built in four layers. The lowest is a task coordinator that maintains the illusion of simultaneity among many cognitive, perceptual, reflexive and environmental processes, on a serial machine. The next layer implements the Phoenix environment itself---the maps of Yellowstone National Park, and the simulations of fires. The third layer contains the definitions of the components of agents---our specific agent design. The fourth layer describes the organization of agents, their communication and authority relationships. Layers 3 and 4 are described in later sections.

The two lowest layers in Phoenix, called the *task coordinator layer* and *map layer* respectively, comprise the Phoenix discrete event simulator. We discuss the task coordinator first. It is responsible for the illusion of simultaneity among the following events and actions:

Fires: Multiple fires can burn simultaneously in Phoenix. Fires are essentially cellular automata that spread according to local environmental conditions, including wind speed and direction, fuel type, humidity, and terrain gradient.

Agents' physical actions: Agents move from one place to another, report what they perceive, and cut fireline.

Agents' "internal" actions: Internal actions include sensing, planning, and reflexive reactions to immediate environmental conditions.

These tasks are not *generated* at the task coordinator level of Phoenix, just scheduled on the cpu there. Fire tasks are generated at the map layer, and agent tasks are generated at the levels described in Sections 5 and 6.

Typically, the task coordinator manages the physical and internal actions of several agents (e.g., one fireboss, four bulldozers, and a couple of watchtowers), and one or more fires. The illusion of continuous, parallel activity on a serial machine is maintained by segregating each process and agent activity into a separate task and executing them in small, discrete time quanta, ensuring that no task ever gets too far ahead or behind the others. The default setting of the synchronization quantum is five minutes, so all tasks

are kept synchronized to within five minutes of each other. The quantum can be increased, which improves the cpu utilization of tasks and makes the testbed run faster, but this increases the simulation-time disparity between tasks, magnifying coordination problems such as communication and knowing the exact state of the world at a particular time. Conversely, decreasing the quantum reduces how "out of synch" processes can be, but increases the running time of the simulation.

The task coordinator manages two types of time: *cpu time* and *simulation time*. CPU time refers to the length of time that processes run on a processor. Simulation time refers to the "time of day" in the simulated environment. Within the predefined time quantum, all simulated parallel processes begin or end at roughly the same simulation time. To exert real-time pressure on the Phoenix planner, every cpu second of "thinking" is followed by K simulation-time minutes of activity in the Phoenix environment. Currently K = 5, but this parameter can be modified to experiment with how the Phoenix planner copes with different degrees of time pressure.

The fire simulator resides at Phoenix's map layer; that is, the map layer generates tasks that, when executed by the task coordinator, produce dynamic forest fires. Phoenix's map, which represents Yellowstone National Park, is a composite of several two dimensional structures, and stores information for each coordinate about ground-cover, elevation, features (roads, rivers, houses, etc.), and fire-state. The fire itself is implemented as a cellular automaton in which each cell at the boundary decides whether to spread to its neighbors, depending on the local conditions just mentioned and global conditions such as wind speed and direction (currently, we do not model local variations in weather conditions). These conditions also determine the probability that the fire will jump fireline and natural boundaries.

The Phoenix discrete event simulation is generic. It can manage any simulations that involve maps and processes. For example, we could replace the forest fire environment with an oil-spill environment. We could replace our map of Yellowstone with oceanographic maps of, say, Prince William Sound. Fire processes have spatial extent, and spread according to wind speed, direction, fuel type, terrain, and so on. They could easily be replaced with oil-slick processes, which also have spatial extent, and spread according to other rules. Similarly, we could replace the definitions of bulldozers and airplanes with definitions of boats and booms.

5. Agent Design, Layer 3.

The third layer of Phoenix is our specific *agent design*, which is constrained by forest fire environment as described in Section 3. For example, because events happen at two dramatically different time scales, we designed an agent with two parallel and nearly-independent mechanisms for generating actions (Figure 6). One generates reflexive actions very quickly---on the order of a few seconds of simulated time---and the other generates plans that may take hours of simulated time to execute. This longer-term planning can be computationally intensive, because it incurs a heavy time penalty for switching contexts when interrupted. For this reason, the cognitive component is designed to do only one thing at a time (unlike sensors, effectors, or reflexes, where multiple activities execute in parallel). Both the cognitive and reflexive component have access to sensors, and both control effectors, as shown in Figure 6.

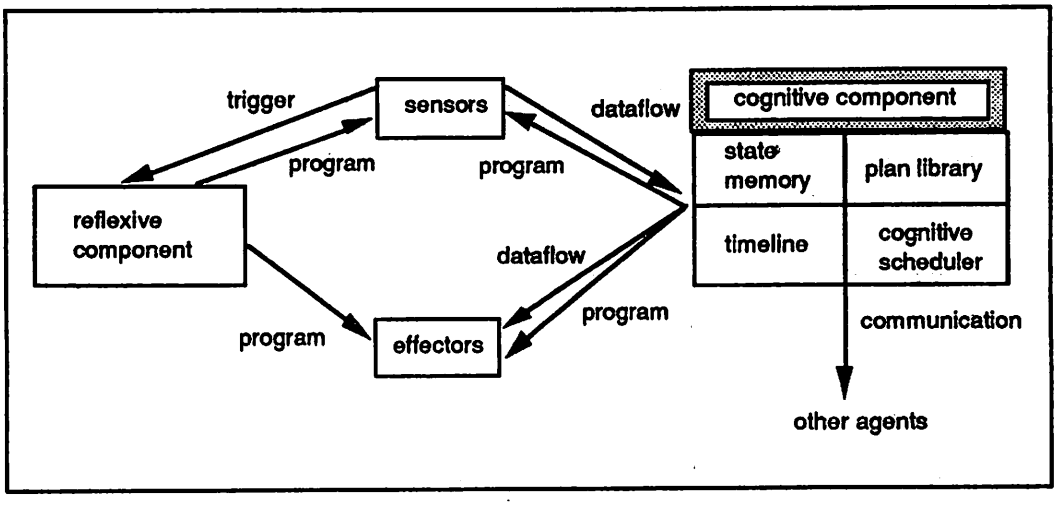

Figure 6: Phoenix agent design.

The agent interacts with its environment through its sensors and effectors, and action is mediated by both the reflexive and the cognitive components. Sensory information may be provided autonomously or may be requested, and sensors' sensitivity may be adjusted by the cognitive component. Effectors produce actions in the world such as information gathering, building fireline, and moving.

Reflexes are triggered by output of sensors. They change the programming of effectors to prevent catastrophes, or they fine tune the operation of effectors. For example, a bulldozer is stopped by a reflex if it is about to move into the fire, and reflexes handle the fine tuning necessary for the bulldozer to follow a road. Reflexes are allotted almost no cpu time, and have no memory of events, so they cannot produce coordinated sequences of actions. They are designed for rapid, unthinking action. Although some researchers have suggested that longer-term plans can *emerge* from compositions of reflexes [1][2], we do not believe that compositions of reflexes can handle temporally-extensive planning tasks such as resource management, or spatially-extensive tasks such as path planning with rendezvous points for several agents. Thus, we have adopted a design in which reflexes handle immediate tasks and a cognitive component handles everything else.

The cognitive component of an agent is responsible for generating and executing plans. Instead of generating plans de novo, as classical hierarchical planners did, the Phoenix cognitive component instantiates and executes stored skeletal plans. We believe this is a good design for the forest fire environment because, first, a relatively small number of skeletal plans is probably sufficient to cope with a wide range of fires; and, second, the store/recompute tradeoff suggests relying on stored plans, rather than computing them, in real-time situations. In addition to controlling sensors and effectors, the cognitive component handles communications with other agents (including integrating sensor reports), and it responds to flags set when reflexes execute. It also engages in a wide range of "internal" actions, including projection (e.g., where will the fire be in 20 minutes?), plan selection and scheduling, plan monitoring, error recovery, and

replanning. Our implementations of some of these capabilities are quite rudimentary, and leave much room for improvement, as we discuss in Section 8.

In overview, this is how the cognitive component works: in response to a situation such as a new fire, an appropriate plan is retrieved from the *plan library* and placed on the *timeline* (Fig. 6). *State memory* stores information, such as weather, resource conditions, and sensory input, that helps the cognitive agent select appropriate plans and instantiate the variables of the chosen plan for the current situation. For example, if the fire is small and nearby, and the weather is calm, then a one-bulldozer plan will be retrieved and instantiated with situation-specific information such as the wind speed and the current location of the fire. The actions in a plan are eventually selected for execution by the *cognitive scheduler*, described shortly. At any time during this process, sensory data may trigger reflexive actions; for example, if the cognitive component is executing a command to move to a destination, and a sensor reports fire ahead, then the reflexive component will send a command to reverse direction. This happens very fast relative to the cycle time of the cognitive component, so the reflexive component sets a flag to tell the cognitive component what it did. When the cognitive component notices the flag, it might modify its plan. The analogy here is to our own reflexes, which yank us away from hot surfaces long before our cognitive apparatus becomes aware of the problem.

With this overview in mind, let us consider the operation of the cognitive component in detail. We will focus on the operation of the *fireboss* agent, which plans the activities of other agents such as bulldozers and crews. Each of these, in turn, plans how to carry out the directives of the fireboss. Because bulldozers and crews have the same architecture as the fireboss (Fig. 6), they can reason in exactly the same way. In the following discussion, we first describe planning when things go according to plan, and then describe error handling, interruptions, and other unexpected events.

When a fire is reported, an *action* called "deal with fire" is retrieved from the plan library and used to create a *timeline entry*, in this case called "deal with fire 27", which is added to the timeline (see Figure 7). Actions are general representations of the cognitive activities the agent can perform, such as path planning or communication, and describe applicability conditions, resource constraints and uninstantiated variables. Creating a timeline entry instantiates an action: binding its variables and adding the temporal constraints that relate it to other actions the agent has chosen to execute. Although timeline entries represent actions, it is not quite accurate to say they are executed (although we will use this terminology where the accurate description is too awkward). In fact, when a timeline entry is created, it inherits a set of *execution methods* from the action it instantiates. Each of these methods will execute the desired action; they differ along dimensions such as the time they require and the quality of their outputs. For example, a single action "plan a path" points to several path-planning algorithms, some which run quickly and return adequate paths, and some that run longer but produce shorter paths. When a timeline entry is selected for execution, the execution method most appropriate to the current circumstances is chosen. By delaying the choice of methods, the cognitive scheduler can reason about its own use of time, and select execution methods that are suited to emerging time constraints.

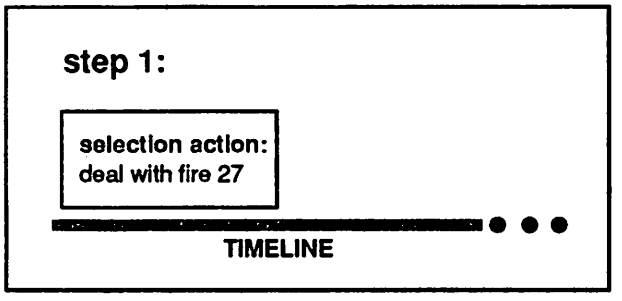

Figure 7: Contents of fireboss's timeline after being notified of a new fire: action to search for a plan to deal with the fire.

If there are entries on the timeline (e.g., "deal with fire 27") then the cognitive scheduler of the Phoenix cognitive component makes three decisions:

Which action to execute next

How much time is available for its execution

What execution method should implement the action

The cognitive scheduler always selects the "next" action on the timeline to execute, but often, several actions have this distinction and a choice must be made. Actions on the timeline may be unordered (and thus equally entitled to "go first") for several reasons: skeletal plans often leave actions unordered so that the cognitive scheduler has flexibility at execution time to select the best order. Or, frequently, the agent is executing several plans simultaneously. This happens, for example, when several fires are reported. The planner formulates plans for each, but doesn't specify temporal constraints among actions from different plans. In the current example, however, the only action on the timeline is "deal with fire 27," so the cognitive scheduler determines how much time is available to execute it and selects an execution method. In this case, it selects a method called *find and filter plan* (step 2, Fig. 8). Its effect, when executed, is to search the plan library for a plan to "deal with fire 27." First it finds all plans for dealing with fires of this type, then it filters the infeasible ones, then selects from the candidates to find the most appropriate one, and lastly, it adds a new action to the timeline called "2 BD surround." (This plan, illustrated in Figs. 2-4, involves sending two bulldozers to a rendezvous point, then to the fire, after which they cut fireline in opposite directions around the fire.)

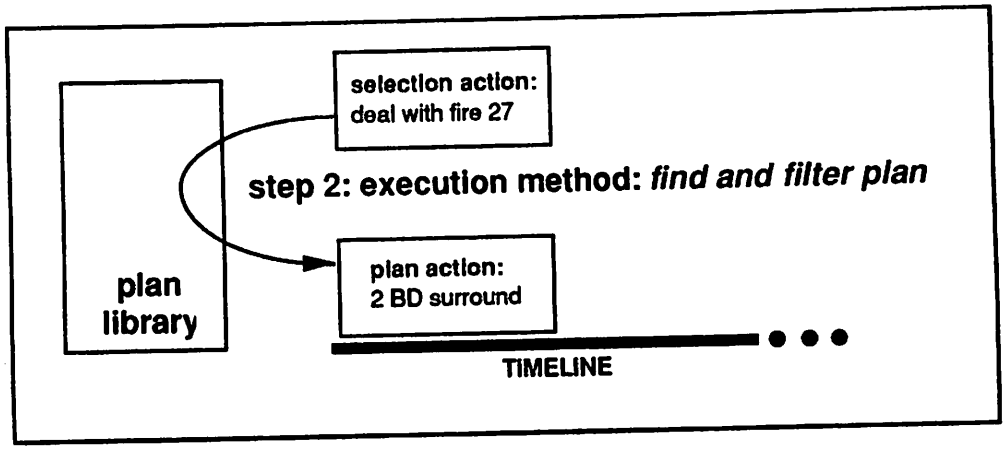

Figure 8: The fireboss executes timeline action, deal with fire 27, which searches the plan library, selects the 2 BD surround plan as appropriate for dealing with new fire, and places the new plan on the timeline.

Once again, the cognitive scheduler selects an action (the only one is "2 BD surround") assesses how much time is available, and selects an execution method. In this case, the method is *expand plan*. The result is to add a network of actions, partially ordered over time, to the timeline (step 3, Fig. 9). The network starts with a placeholder action, s, followed by two unordered actions that allocate bulldozers 1 and 2, respectively. The next action determines the rendezvous point for the bulldozers. Then two unordered actions bind the variables in the plan with the current wind direction and the previously-determined rendezvous point. Space precludes showing the rest of the plan in Figure 9.

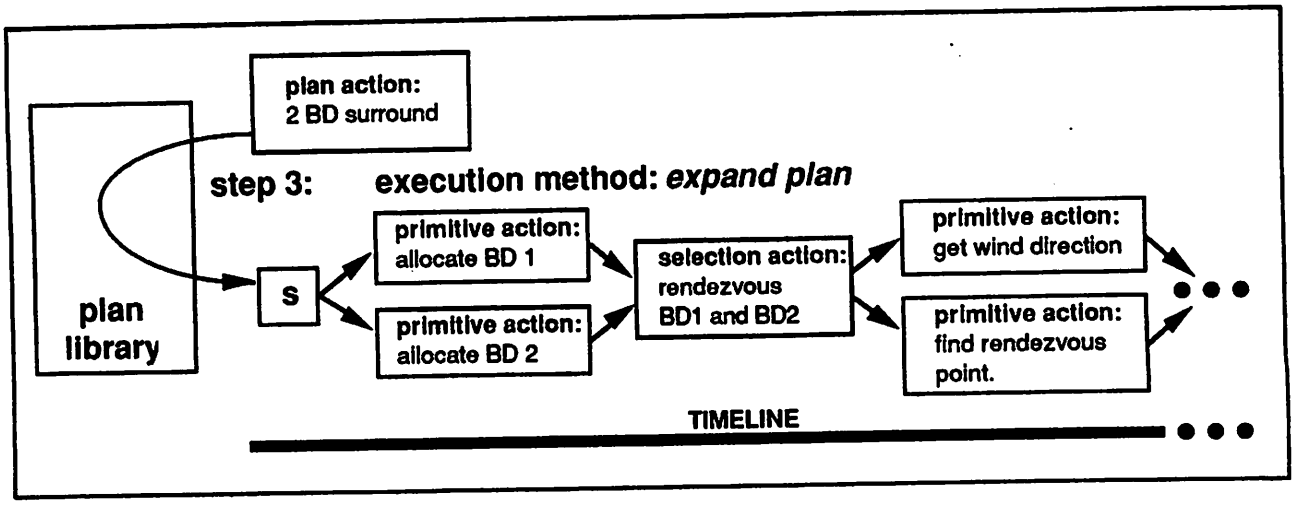

Figure 9: The fireboss executes timeline action, 2 BD surround, which expands into a network of plan steps.

The cognitive scheduler again looks at the timeline, and now must make a decision about which action to select. The "allocate bulldozer" actions are unordered, so one must be selected to go first. Then, as before, the cognitive scheduler assesses the available time

and selects an execution method. We will leave this example here, and discuss it further in Section 7.

Three kinds of actions can be differentiated by their effects on the timeline when they are executed: *selection* actions, like "deal with fire 27" result in a search of the plan library, after which a *plan action* such as "2 BD surround" is posted on the timeline. Plan actions are placeholders for plans; executing them results in plan expansions being posted on the timeline. Many of the actions in a plan are of the third type: *primitive* actions that result in a computation (e.g., calculating a route), or a command to a sensor or effector. But a plan can contain any of the three types of actions; for example, the expansion of "2 BD surround" contains a selection action. When executed, it will result in a search of the plan library for a plan to rendezvous the two bulldozers. Plans can also contain plan actions, which, when executed, add subplans to the network. This is our mechanism for representing hierarchical plans. Lastly, plans may contain just a single, primitive action, such as finding the rendezvous point for two bulldozers.

We have discussed how actions are scheduled and executed when everything goes according to plan, but in the Phoenix environment it rarely does. Phoenix agents have three abilities, all rudimentary, to handle unexpected events. Reflexes, operating on a very short time scale, can halt or modify potentially injurious actions, such as straying into the fire. By design, reflexes do very little processing, and return very little information. When a reflex halts a bulldozer, it simply posts a flag for the cognitive component; it does not interrupt the cognitive component to explain what it did. The cognitive component doesn't become aware of the change until it executes a regularly-scheduled status-checking action. In fact, by design, nothing ever interrupts a cognitive action. This is because the cost of saving state and switching context is prohibitive. Instead, the reflexive component of a Phoenix agent is expected to deal with situations as they arise. Most, like staying parallel to a moving fire, will never require the attention of the cognitive component anyway; but even when a serious problem comes up, the reflexive component is designed to keep the agent functioning until the cognitive component finishes its current task.

The second mechanism for handling unexpected situations is error recovery and replanning. Errors are unexpected events that preclude completion of an action or a plan. For example, bulldozers will travel to their designated destinations but fail to find a fire, path planning will sometimes fail to generate a path, selection actions will search the plan library but fail to find a plan that satisfies all constraints, and so on. Currently, over a dozen types of error can arise in Phoenix, although we don't have plans to deal with them all yet. The error handling mechanism is to post on the timeline a "deal with error" selection action, which, when executed, generates a plan for dealing with the error. Currently, error recovery involves very little tinkering with the actions that are currently on the timeline, that is, no serious replanning.

Lastly, Phoenix agents have limited abilities to monitor their own progress. This is accomplished by generating expectations of progress, and matching to them actual progress. In the near future, this mechanism (called *envelopes*, Sec. 8) will enable Phoenix cognitive components to predict failures before they occur.

In sum, planning is accomplished by adding a selection action to the timeline to search for a plan to address some conditions. Executing the selection action places an appropriate plan action or primitive action on the timeline. If this new entry is a plan action, 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 a primitive action, execution instantiates the requisite variables, selects an execution method, and executes it. In general, a cognitive agent will interleave actions from the several plans it is working on.

This style of planning is "lazy skeletal refinement"---lazy because some decisions are deferred until execution time. Specifically, plans are not selected until selection actions are executed, and execution methods are selected only when an action is about to execute. This style of planning and acting is designed to be responsive to a complex dynamic world by postponing decisions, while also grounding potential actions in a framework (a skeletal plan) that accounts for data, temporal and resource interactions. The combination of a reflexive and cognitive component is designed to handle time scale mismatches inherent in an environment that requires micro actions (e.g., following a road) and contemplative processing such as route planning, which involves long search times and integration of disparate data. We must stress, however, that Phoenix is too early in its development to claim that our agent design is *necessarily* the best one for the Phoenix environment (see Sec. 8).

6. The Organization of Fire-Fighting Agents in Phoenix

The fourth layer of the Phoenix system is the centralized, hierarchical organization of fire fighting agents. Because all agents have the same architecture, many other organizations of agents are possible. Our centralized model is neither robust (e.g., what happens if the fireboss is disabled?) nor particularly sophisticated. But it is simple, a great advantage in these initial phases of the project. One fireboss coordinates all fire fighting agents' activities, sending action directives and receiving status reports, including fire sightings, position updates, and actions completed. The fireboss maintains a global view of the fire situation based on these reports, using it to choose global plans from its plan library. It communicates the actions in these plans to its agents, which then select plans from their own plan libraries to effect the specified actions. Once their plans are set in motion, agents report progress to the fireboss, from which the execution of global plans is monitored. All communication in this centralized implementation is between the fireboss and individual agents - there is no cross-talk among the agents.

The fireboss maintains global coherence, coordinating the available fire fighting resources to effectively control the fire. It is responsible for all the work required to coordinate agents, such as calculating rendezvous points, deciding how to deploy available resources, and noticing when the fire is completely encircled with fireline. The plans in its plan library are indexed by global factors, such as the size of the fire and the weather conditions. The actions in its plans are mostly concerned with coordinating and directing other agents. The fireboss' state memory records the current environmental conditions, where agents have seen fire, what actions have been taken, what agents are available, and how well global plans are progressing. The fireboss is currently implemented without any sensors, effectors, or reflexes. It is a cognitive agent that relies

solely on communication for its knowledge of what develops in the outside world, although it does have a map of the static features of Yellowstone.

Each of the other fire fighting agents has a local view of the environment based on its own sensory input. They have access to maps of the static features in Yellowstone such as ground cover, roads, and rivers, but only know about dynamic processes such as the fire from what they see or are told by the fireboss. Sensors have a limited radius of view, though agents are able to remember what has been perceived but is no longer in view. The fireboss's global view is available to an agent only through communication. A bulldozer is an example of an agent type. It has a movement effector that can follow roads or travel cross-country. When it lowers its blade while moving, it digs fireline and moves more slowly. It has a sensor that sees fire within a radius of 512 meters. Another sensor picks up the contour of a fire (within its radius of view). When a bulldozer is building fireline at the contour, it uses the follow-fire sensor in combination with the movement effector (with lowered blade) and a reflexive action that helps maintain a course parallel to the contour. As the contour changes, the contour sensor registers the change, which triggers a reflex to adjust the movement effector's course. The bulldozer's plan library has plans for simple bulldozer tasks such as following a given path or encircling a fire with fireline.

Although all agents have the same architecture (i.e., timeline, cognitive scheduler, plan library, state memory, sensors, effectors, and reflexes) they do not have the same plans, reflexes, sensors or effectors. The difference between the fireboss and other agents lies in their views of the world and the types of plans each knows. The lines of authority and division of responsibilities are clear; the fireboss maintains the global picture, based on the local views of its agents, and it executes plans whose effects are to gather information, send directives to agents, and coordinate their activity via communications. In contrast, the agents execute 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" whose sensors and effectors are other agents.

7. An Example

We now return to the example that we introduced in Section 2 and used in Section 5 to illustrate cognitive scheduling. In this *two bulldozer surround* plan, the fireboss instructs two bulldozers to rendezvous, then go to the fire and build fireline around it in opposite directions. Figures 2, 3, and 4 show the progress of this plan. Each offers two views of the situation. Figure 2 shows the real world in the left pane, and the fireboss's view in the right pane. Note that the fireboss is not yet aware of the fire. What it knows about new fires is based on status reports from a watchtower agent (not shown). Each watchtower has a sensor programmed to look for new fires at regular time intervals. When the watchtower spots this fire, it reports the location and size to the fireboss. Based on this report and the resources available, the fireboss selects the two bulldozer surround plan. The first plan steps allocate the bulldozers, which ensures they are not busy with other tasks and assigns them to this plan. The next step instructs them to rendezvous so they can follow the same route to the fire. While they rendezvous, the fireboss locates the rear of the fire (the upwind side), and calculates a route to the fire that approaches it from that

direction. The next two steps communicate to each bulldozer instructions to follow the given path and encircle the fire. They are given clockwise and counterclockwise encircling directions, respectively.

After receiving its instructions, each bulldozer searches its plan library to find a plan for following the path and encircling the fire in the given direction until it closes the fireline. Neither bulldozer knows about the other, nor does either know the full extent or precise location of the fire. Recall that the fireboss doesn't know exactly where the fire is, either, so the path it supplied to the bulldozers may direct them wide of the fire, or, more often, to a location that is burning. In this example, the path given by the fireboss ends in the fire, so the bulldozers will follow the path until they detect it. In Figure 3 we see the bulldozers starting to build line. In the fireboss view (right pane) each one appears at the position it had reached when it made its last status report. Thus they are at slightly different positions that are out of date with respect to their real positions in the left hand pane.

When fire is seen, a bulldozer reflex is triggered to stop its movement effector. A cognitive action also notes that the sensor has seen fire and reprograms the sensors and effectors with the right combination of instructions to follow the fire in the direction specified by the fireboss, building fireline as it goes. A message is sent to the fireboss to signal the start of line-building. The bulldozer will continue to build line until instructed to stop by the fireboss. In Figure 4 we see in the left pane that the bulldozers have almost encircled the fire. In the right pane is the view of the bulldozer encircling in the clockwise direction. Note that it only knows about the fire it has seen as it was building line. It is just coming within range of the other bulldozer (see the spot of fireline to its northeast).

This simple bulldozer plan, to follow a path and encircle a fire without reference to other bulldozers, can be used by one, two, or many bulldozers. The fireboss, with its global view, picks points around the fire, selects any number of bulldozers, and directs each to go to one of the points and build fireline in a specified direction. The bulldozers act only with regard to their instructions and the local information in their field-of-view. If the bulldozers fail to fully encircle the fire (for whatever reason), the fireboss is responsible for noticing the failure, based on what is reported to it from watchtowers and bulldozers.

Figure 10 shows the state of the fireboss's timeline as the bulldozers are closing off the fireline (see Figure 4). The network in the top left box is the top level of the timeline, which contains four entries and reads from left to right. There is a startup-action and an end-action (place-holders), and two entries with no temporal constraint between them. The top entry is an action that executes periodically and updates state memory with new information about the environment. The bottom entry is an action that is placed on the fireboss's timeline automatically by the report of a new fire. It causes a plan to be selected from the fireboss's plan library, based on the characteristics of the reported fire, and then expanded on the timeline, as illustrated in Section 5. The selected plan is shown in the top right box, and its expansion is shown in the two lower boxes (the plan unfolds left-to-right, and is continued in the lowest box). Entries preceding the shaded one have already been executed. These include allocating each bulldozer, instructing them to rendezvous, calculating a route for them to take to the fire, and (in undetermined order) instructing them to follow that route and encircle the fire.

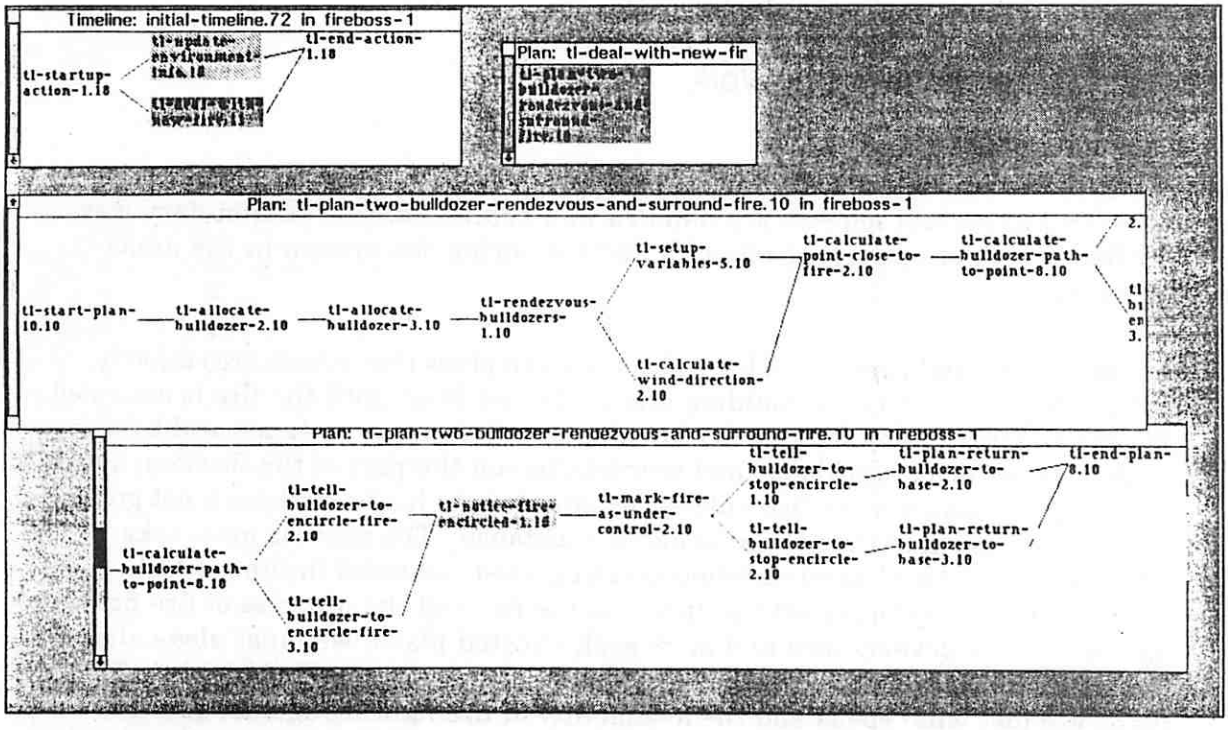

Figure 10: The fireboss's timeline

This figure shows the fireboss's timeline, including plan expansions, as viewed from the Phoenix desktop (headers for boxes name parent nodes). The box at the top left is the top level of the fireboss's timeline. The darkly shaded entry here is the action to deal with a newly sighted fire. Dark shading means an entry is executing or has a child node that is executing. The fireboss selected the **two-bulldozer-rendezvous-and-surround-fire** *plan, which is shown in the top right box. The expansion of this plan is shown in the lower boxes. It starts with the timeline entry* **tl-start-plan-10.10**, *reads from left to right and wraps around into the lower box. Entries are ordered in temporal sequence; a split after an entry represents subsequent entries with no temporal constraints between them. The current entry (shaded) is executed by periodically checking to see whether the fire is completely encircled.*

The fireboss is currently waiting for the bulldozers to finish encircling the fire (see shaded entry in lowest box). The execution method for this entry runs periodically, checking to see if there is continuous fireline surrounding the fire. Once this is true, the bulldozers will be instructed to stop building fireline. This is necessary because the fireboss maintains the global view of the fire, and must tell the bulldozers when the fire is surrounded. (The bulldozers' timelines are not shown.) Once each bulldozer has been instructed to stop building line, a plan will be selected for returning it to the bulldozer base.

8. Current Status and Future Work

The Phoenix system is very much a work in progress. As is clear from the preceding sections, several important aspects are handled in a rudimentary or preliminary way. Currently five people are pursuing research and enhancing the system in the areas described below:

More sophisticated plans: We have about a dozen plans that attack fires directly, with up to four bulldozers, building line at the fire front until the fire is encircled. We are starting to develop *indirect attack* plans that incorporate natural barriers. This requires more knowledge and coordination on the part of the fireboss; since other agents can't see the fire unless they are close to it, the fireboss must guide their activities when they are working at a distance. The fireboss must take advantage of natural barriers when deciding where to build fireline, which requires the ability to project the spread of the fire and the progress of fire fighting agents. As we develop new and more sophisticated plans, we must also enhance the mechanisms by which agents select plans. Currently, the keys for selecting plans are just wind speed and the availability of fire-fighting agents; as well as some plan-specific criteria such as whether bulldozers are nearby or distant when the plan is selected. The keys will have to become more discriminating, and we will probably have to develop more sophisticated plan-selection mechanisms.

Monitoring: We have designed a general monitoring mechanism called *envelopes* that minimizes the cognitive resources devoted to monitoring while providing early warning of plan failure. Envelopes incorporate expectations of how plans are to proceed; they represent these expectations functionally. As actual progress is reported, it is compared with these expectations, and deviations outside certain parameterized thresholds are flagged for cognitive attention. For example, 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 agent's progress. The envelope 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 envelop signals the planner that the deadline is in jeopardy,

providing an early warning of failure. Currently, we have hooks for envelopes in plans, but we do not have the mechanisms to replan when envelopes are violated.

Error recovery and replanning: These activities are implemented as cognitive actions, just like plan selection and plan expansion. When an error is detected in a plan, an action is posted to the timeline that inspects the error and attempts to fix the existing plan. Consider, for example, a failure on the part of a bulldozer to find fire at the location to which it was sent. A plan we currently have to fix the error is to travel a little further in a specified direction, looking for the fire. A really intelligent error recovery will know when to try cheap fixes, such as modifying a destination; and when to begin a search for a way to significantly modify a plan (e.g., by dispatching another bulldozer); and when, as a last resort, to abandon the current plan and begin from scratch. Error recovery and replanning will depend significantly on intelligent monitoring; in fact, envelopes are designed to predict errors before they happen, minimizing "downtime."

Cognitive scheduling: We need to enhance the scheduling abilities of the cognitive component to make agents responsive to real-time demands in fire fighting. This is particularly true for the fireboss in our implementation, since it is essentially a cognitive agent. Currently, scheduling involves three actions: selecting an action to execute, deciding how much time is available, and selecting an execution method. But although these actions are "charged" for the time they use, they are not themselves scheduled, nor are there multiple execution methods to implement them. In short, the cognitive scheduler is a separate "interpreter" of the timeline. To make the scheduling of actions completely uniform, scheduling actions must themselves be scheduled. In addition, we must develop scheduling strategies, along the lines suggested in Lesser, Durfee, and Pavlin's approximate processing proposal [8].

Agent Architecture. To facilitate experiments with different agent designs in different environments, we have started to build a generic agent architecture. It is a collection of parameterizable structures that represent the design of parts of an agent. For example, our generic action structure includes pointers to execution methods, to envelopes, and to predicates that are tested before the action is selected. Generic execution methods, in turn, contain estimates of their time requirements, their prerequisites, and on. We also have generic structures for sensors and effectors. In the near future, we will implement generic structures for strategies, including memory access strategies and cognitive scheduling strategies. The eventual goal is a full generic agent architecture that makes it easy to implement different agent designs by specifying how the agent manages its sensors and effectors, how it manages its memory, and how it decides what to do next.

Organization and communication: We have demonstrated one way to organize a multi-agent planner in the Phoenix testbed, but the agent architecture certainly supports others. Work is underway in Victor Lesser's lab to build a cooperating, distributed planner for the Phoenix testbed. Although preliminary, this model assumes multiple firebosses, each with *spheres of influence* (geographic areas and agents) under its control, who cooperatively fight fires at their borders, loaning

resources to neighbors, or redrawing their boundaries to shift the work load in times of stress. While this model is similar to the Phoenix planner in the relationship between firebosses and agents, it adds a cooperative relationship between firebosses.

Learning: Phoenix agents should learn to improve their performance. The opportunities for learning are myriad: we can learn new reflexes, and chain reflexes together to learn short plan fragments. We can learn new plans from patches to failed ones. We can learn correlations between environmental conditions, such as changes in wind direction, and failures, such as bulldozers becoming trapped in the fire. Currently, we are extending the error recovery mechanisms to learn patches to failed plans. This is one aspect of Adele Howe's dissertation work [7]. Allen Newell recently pointed out that "you can't program SOAR" because much of its behavior emerges from sequences of locally-selected chunks, and there is really no way to predict how a chunk, added by hand, will make the system behave. We have found the same to be true of actions and reflexes in Phoenix, and concurr with Newell that once a system attains a degree of complexity, it must learn to improve its performance itself.

9. Conclusion.

The development of Phoenix has been intimately tied to our evolving ideas about AI research methodology, and specifically to our understanding of the role of evaluation in AI research [5][6] . Clearly, the evaluation of Phoenix must be with respect to the goals of the project. Moreover, it must tell us not only whether we have succeeded, but whether we are succeeding; and why, or why not. The goals of Phoenix are, as noted in Section 1, of three kinds. Our technical goals are to build a real-time planner with learning, approximate scheduling, envelopes, and the other features noted above. Our scientific goal is to understand how environmental characteristics influence agent design---the relationships discussed in the context of the behavioral ecology triangle (Fig. 5). Lastly, we are using Phoenix as a framework in which to develop AI methodolgy.

Progress toward each of these goals is evaluated differently. Phoenix is parameterized and instrumented at all its layers to facilitate evaluations of specific technical developments; for example, we can assess whether an approximate scheduling algorithm is robust against varying time pressure because we can vary time pressure while holding other factors constant. We can run fire scenarios in dozens of conditions, with dozens of variations in the algorithms used by the Phoenix planner. These experiments are scheduled to begin in the Fall of 1989. They will enable us to demonstrate the utility of our technical solutions, explain why they are solutions, and discover the limits on their scope [5].

But clearly, these cannot be the only aims of the experiments. While it is valuable to probe the scope and efficacy of specific techniques, such experiments will not necessarily address our scientific goals. We might show that a Phoenix planner works well in the Phoenix environment, but not how the environment constrains the design of planners. Furthermore, unless we are trying to answer specific questions of this sort, experiments with techniques will be unguided. There are dozens of variations on the Phoenix planner,

and hundreds of environmental conditions in which they might be tested. To guide the search of this space, we will generate and test general rules that justify and explain the design of agents. These rules will call upon functional relationships that capture tradeoffs. For example, the well known store-recompute tradeoff lurks in the design of the Phoenix planner: we use it to justify the decision to rely on stored plans in an environment that exerts time pressure, favoring storage over computation. Perhaps there is a general rule here (e.g., under time pressure, rely on storage over computation), or perhaps there are many specific variants of this rule, for environments with different kinds of time pressures and agents with different kinds of store-recompute tradeoffs. In any case, our scientific goal is to discover functional relationships (and to exploit those we already know, like the store-recompute tradeoff), and to embed them in rules for designing intelligent agents. To evaluate progress, we need to measure not the performance of the agents, but the extent to which that performance can be predicted. If we really understand the relationships between environment characteristics, agents' behaviors, and agents' designs, then we should be able to predict that agents with particular designs will behave in particular ways under particular environmental conditions.

Although we are far from this goal, it is paradigmatic of the style of AI research we advocate. To evaluate the success of this methodological stance will take a long time, but if it is possible, there is surely no better aim for AI than to understand---to the point of being able to predict behavior---how to design intelligent agents in complex environments.

References

1. Agre, P. E. and D. Chapman. Pengi: An Implementation of a Theory of Activity. Proceedings of the Sixth National Conference on Artificial Intelligence. 268-272, 1987.

2. Brooks, R. A. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation. RA-2(1). pp. 14 -23. 1986.

3. Cohen, P. R., Greenberg, M. L., Hart, D. and Howe, A. E. An Introduction to Phoenix, the EKSL Fire-Fighting System. EKSL Technical Report. Department of Computer and Information Science. University of Massachusetts, Amherst. 1989.

4. Cohen, P. R. Why knowledge systems research is in trouble and what we can do about it. EKSL Technical Report. Department of Computer and Information Science. University of Massachusetts, Amherst. 1989.

5. Cohen, P. R. and A. E. Howe. How Evaluation Guides AI Research. AI Magazine. 9(4): 35--43, 1988

6. Cohen, P. R. and A. E. Howe. Towards AI Research Methodology: Three Case Studies in Evaluation. IEEE Transactions on Systems, Man, and Cybernetics, to appear, 1989.

7. A. E. Howe. Adapting Planning to Complex Environments. Dissertation proposal, 1989.

8. Lesser, V. R., E. H. Durfee and J. Pavlin. Approximate Processing in Real-Time Problem Solving. AI Magazine. : 49-61, 1988.