Addressing Real-Time Constraints in the Design of Autonomous Agents

Adele E. Howe, David M. Hart, Paul R. Cohen

COINS Technical Report 90-06

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

Abstract

The Phoenix project is an experiment in the design of autonomous agents for a complex environment. The project consists of a simulator of the environment, a basic agent architecture, and specific implementation of agents based on real-time techniques; the first two parts have been constructed, the third is on-going. The facets of Phoenix that facilitate real-time research are: a simulator parameterized for varying environmental conditions and instrumented to record behaviors, an agent architecture designed to support adaptable planning and scheduling, and methods for reasoning about real-time constraints.

---

[0]This research has been supported by DARPA, # F306 02-85-C-0014; the Office of Naval Research, under a University Research Initiative grant, N00014-86-K-0764; the Office of Naval Research, # N00014-88-K0009, and a grant from the Digital Equipment Corporation. We wish to thank Mike Greenberg for his keen understanding of design issues and mastery of programming that made Phoenix what it is today. We also wish to thank Paul Silvey and David Westbrook for their help.

[0]To appear in the *Real-Time Systems Journal*

1 Introduction

Planning research in Artificial Intelligence is experiencing a 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 time and uncertainty. Time and uncertainty are interacting phenomena because we are often uncertain about the future—that is, about how the environment will change over time—and because we often introduce uncertainty when, under time pressure, we accept approximations and estimates. Time pressure arises in environments that change continuously in ways beyond the control of the agents, producing unanticipated problems at unanticipated rates. Real-time planning, in these environments, requires keeping up with the changing conditions. Realistically, this means that the time required to decide what to do must be less than the time taken by the environment to effect changes that render the intended action inappropriate or unexecutable. We think of real-time planning as managing scarce temporal and physical resources. These resources constrain the time and information available to support decision making.

The goal of the Phoenix project is to understand the design of intelligent agents that interact in an environment in which the success of agents depends on their ability to cope with uncertainty in a timely way. Our general research goal is to design, explain

and predict behavior of intelligent agents in complex environments, environments in which time pressure is one of many contributors to complexity in the environment. A recent paper[2] describes the Phoenix project with respect to this general goal and as such, provides more detail on the methodology underlying the project and the agent architecture. In contrast, this paper focuses on the real time characteristics of the environment and emphasizes how our research addresses those characteristics.

The Phoenix project has two principal components: a simulator and an agent architecture. The simulator provides the environment in which to test our agent designs. The agent architecture embodies our ideas on agent design and provides a structure for embedding methods of planning and problem solving. This paper will describe the design and the present level of implementation for both of the components (in Sections 2 and 3, respectively), explaining those design decisions justified by real-time considerations. Section 4 will describe specific methods being added to the agent architecture to address real-time control in the agent architecture.

2 The Environment

The simulator provides the environment for our autonomous agents[1]. This section describes a real-time application domain, our implementation of that domain in the Phoenix simulator, and the facets of the simulator that facilitate experimentation.

2.1 The Phoenix Domain

The Phoenix task is to control simulated forest fires by deploying simulated bulldozers, crews, airplanes, and other objects. Forest fires spread in irregular shapes, at variable rates, determined by ground cover, elevation, moisture content, wind speed

[1]For some researchers, the use of a simulated environment, rather than a real environment, is fundamentally flawed. Yet, the Phoenix simulator includes characteristics that make "real" environments demanding, while also facilitating experimentation, both toward understanding the limitations and capabilities of techniques and comparing related techniques. We discuss the merits of simulators in more detail in [2].

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 boundaries.

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.

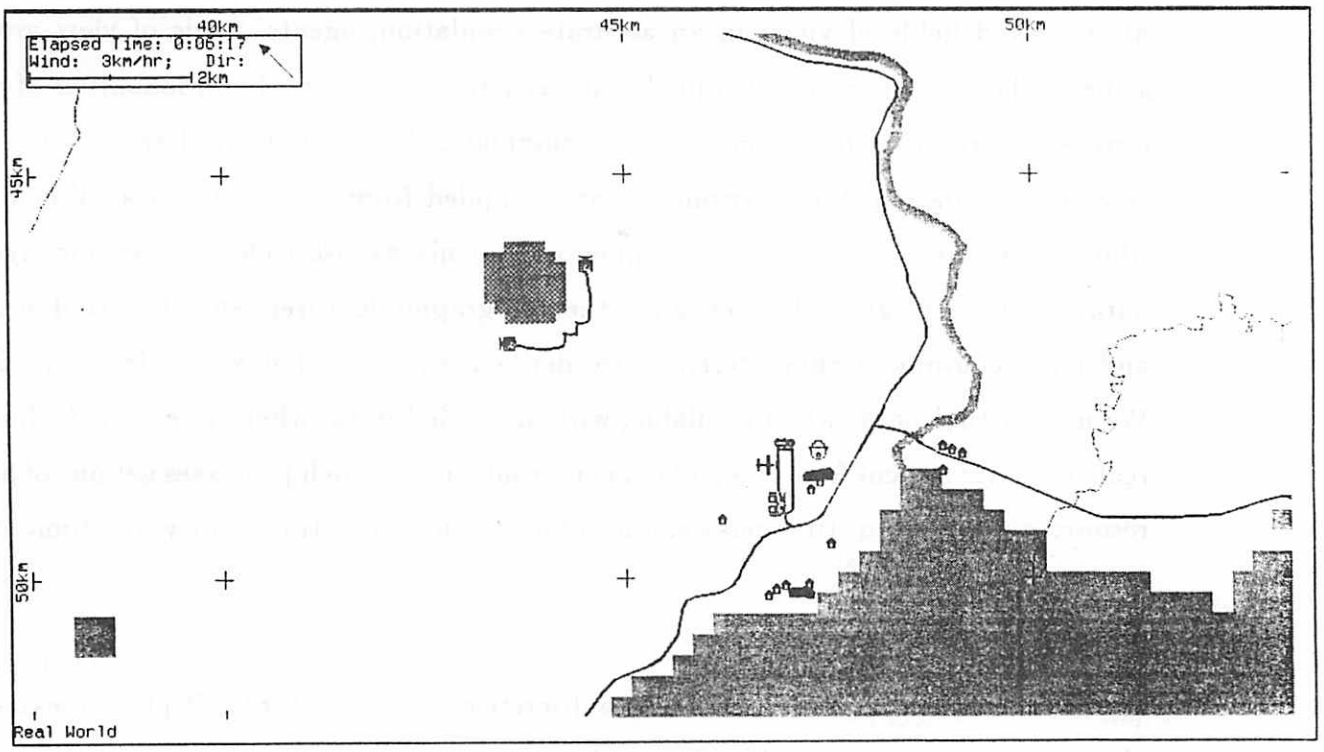

Figure 1: View of Yellowstone from Phoenix Simulator

The Phoenix simulator's environment is 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[2] The grey region at the bottom of the screen is the

[2]The system usually executes with a color display, but for purposes of reproduction, this figure was

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 Smokey the Bear symbol in the bottom left corner marks the location of the fireboss, the agent that directs and coordinates all others. Two bulldozers are shown cutting fireline around a fire in this figure. Two other bulldozers are parked near the fireboss, along with a plane and a fuel carrier.

Phoenix is a *realistic* simulator of forest fires and forest fire fighting. It is important to distinguish realism and accuracy. Realism is necessary for our research goals; accuracy is not. Here are some examples of the distinction: In our realistic simulation, processes become uncontrollable after a period of time; in an accurate simulation, the period of time is the same as it is in the real world. In our realistic simulation, agents have limited fields of view; in an accurate simulation, agents' fields of view are the same as they are in the real world. In our realistic simulation, the probabilities of environmental events such as wind shifts are summarized by statistical distributions; in an accurate simulation, the distributions are compiled from real-world data. When possible, we use accurate data; for example, in Phoenix we use Defense Mapping Agency data of elevation, ground cover, and other geographic features, and the fire dynamics and basic equipment characteristics are derived from U.S. Forest Service manuals[8]. We used actual data, when available, with simplified data, when necessary, to build a realistic environment for our agents, an environment in which processes get out of hand, resources are limited, time passes, and information is sometimes noisy and limited.

2.2 Implementation

The Phoenix simulator is based on a discrete event simulator (DES) that creates the illusion of a continuous world, where natural processes and agents are acting in parallel, on serial hardware[3]. In the simulation, fires burn continuously over time, affected by changing environmental conditions such as wind and humidity. As the fire spreads,

captured from a black and white display.

[3]Phoenix is implemented on a Texas Instruments Explorer II Lisp Machine.

agents act in concert to control it. Some of these actions are physical, as in digging fireline and cutting trees. In parallel with these physical actions, agents are perceiving, moving, reacting to perceived stimuli, and thinking ahead about what action(s) to execute next.

To simulate the parallelism in the environment, agent and environment activities are segregated into separate tasks. These tasks are then executed in small, discrete time quanta, ensuring that no one task gets too far out of synch of the others. By default, the synchronization quantum is set to five minutes of simulation time. So the *maximum* time separation between environmental processes is five minutes; however, most processes will be more closely matched. Changing the synchronization quantum affects the efficiency of the simulator and the realism of the state of the world. Decreasing the quantum allows the processes to be more tightly synchronized and so better integrated temporally, but exacts an efficiency cost in terms of cpu utilization and task swapping. Increasing the quantum improves the cpu utilization and so makes the simulator run faster, but increases the time disparity between tasks, magnifying coordination problems such as communication and information about the state of the environment.

To control the synchronization of the processes, the DES manages two types of time: cpu time and simulation time. CPU time is the amount of real world processing time consumed by the processes. Simulation time is the amount of environment time that passes while acting or thinking, the "time of day" in the simulated environment. Types of tasks differ in how they are "charged for" cpu time and simulation time. Sensory tasks run for very short intervals of simulation time, after which they are rescheduled; this gives them a high sampling rate compared to the rate at which the world is changing. Effector tasks may use very little simulation time, or the full synchronization quantum. Fire tasks always run for the full synchronization quantum. All are allotted as much cpu time as they need by the task coordinator; there is no constant proportionality between the simulation time and the cpu time they require. In contrast, to exert real-time pressure on the Phoenix planner, every cpu second of cognitive activity is followed

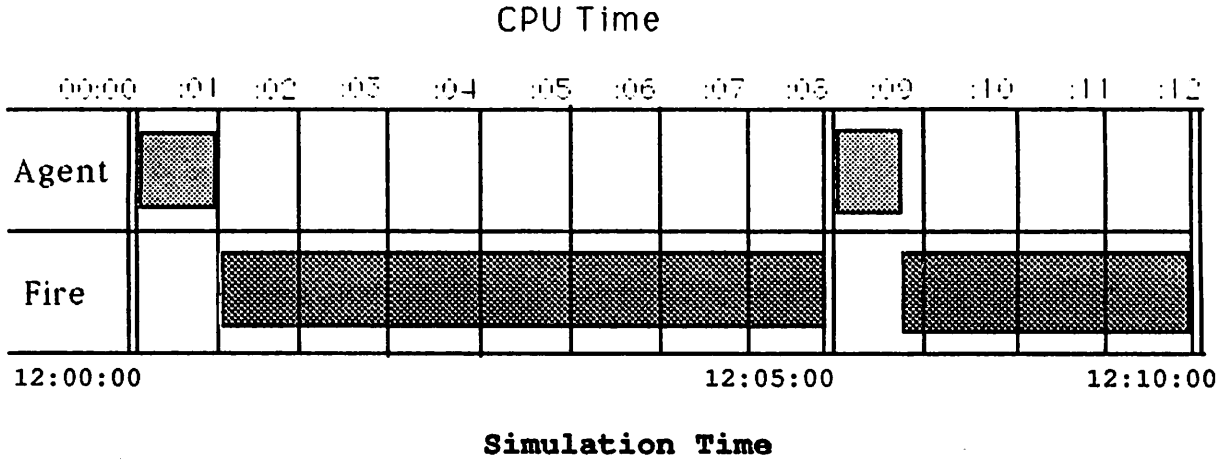

Figure 2: How simulation time corresponds to real time for two types of processes

by K simulation-time minutes of activity in the Phoenix environment (currently K = 5, but it can be adjusted); we refer to this constant K as the real-time control knob.

Imagine it is now 12:00:00 in the simulated world, and an agent is about to begin planning. After one cpu second, simulation time for the agent is 12:05:00. The fire is thus "owed" five minutes of simulation time. It may take 7 cpu seconds to calculate the effects of five minutes of fire. Moreover, simulation time is still 12:05:00, because the agent and the fire are simulated parallel processes. So after eight cpu seconds (one for the planner and seven for the fire), we have simulated five minutes of planning and five minutes of fire, and both processes are paused at 12:05:00. Figure 2 shows the relationship of the agent and fire processes to real and simulation times. Cognitive tasks are allotted a full synchronization quantum each time they run. At times there are not enough cognitive activities to fill a quantum (as in the second time the agent process runs in Figure 2), in which case the task ends and waits to be rescheduled. Some cognitive activities take longer than a full quantum, in which case their internal state is saved between quantum steps.

2.3 How the simulator facilitates research in real-time techniques

By controlling the domain features and the allocation of time through the simulator, we can vary the effect of the environment on the agent interacting in it. The domain features change environmental conditions and can be set as parameters of the simulator. The most obvious examples are weather conditions, e.g., wind speed, wind direction and lightning strikes. These parameters may be programmed to change at designated times as part of a scenario, thus defining baselines for comparison purposes.

To ensure that we will always be able to assess performance under time pressure, irrespective of the speed of problem solving algorithms and their supporting code, the Phoenix simulator allows us to control the allocation of time to the agent's thinking with the 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 cognitive activities 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 under temporal constraints. 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, an agent should still be able to cope. Real-time planning must be robust against small changes in the ratio of thinking time to world time.

Evaluating behavior under changing environmental conditions requires control over those conditions and observable behavior. The simulation parameters offer the control over the environment. System instrumentation provides the metrics for behavioral evaluation.

Real-time performance can be instrumented on three levels[4]: low level, middle level, and high level. Low level metrics are largely hardware dependent estimates of how the software system is utilizing the hardware. With low level metering turned on, the system collects disk accesses and run times. For selected tasks (e.g., the fire simulation and each of the agents), metering collects data on number of times each function is called, its average run time, and cumulative run time.

Middle level metrics are mostly specific to the agent architecture; they are metrics that evaluate how the software structure is performing in the environment. Consequently, metering at this level depends on the design of the agents.. For example, a blackboard solution may call for metering of hypotheses maintained, whereas this metric may be meaningless to another type of solution. Middle level metrics that are not specific to a particular architecture assess the interaction between the agent and the environment (e.g., communication overhead, response times to environmental changes, rate of action failure, and number of errors generated) and the ability of scheduling algorithms to fulfill deadlines. Some examples of metrics for scheduling are the number of tasks that miss deadlines [11], the success ratio of percentage of tasks that meet deadlines weighted by their importance [11] and the average time delay between when a task is eligible and when it is executed [10]. We are in the process of defining and implementing this level of metrics.

High level metrics are domain specific. These metrics record the features of the environment that are affected by the agents: the destruction produced by uncontrolled fires and the resources consumed by the agents. Fire destruction is measured by amount and type of forest, houses, and agents burned. Resource allocation is measured by amount and type of agents employed to fight the fire, gasoline consumed, fireline cut, distance traveled, and time required to contain the fire.

[4]The three level characterization was suggested by Nort Fowler in personal communication.

3 Design of Autonomous Agents

A uniform agent architecture is shared by all agents. This architecture is the *structure* of the agent, the "hardware" that dictates the fundamental faculties and limitations of the agent. The structure endows and bounds acuity, speed of response, and breadth of action; it constrains what an agent can do, but not what it does. Specific methods *control* what the agent does. These methods determine what to do and how to do it. This dichotomy between structure and control is reflected in this section and the one following it, agent architecture and techniques for real-time problem solving. This section describes the structure, a general agent architecture that provides the basic capabilities needed to respond to the demands of the environment. The following section describes techniques that control the structure and so determine what the agent does in response to real-time pressures.

3.1 Phoenix Agent Architecture

The agent architecture has four components[5] (see Figure 3). Sensors perceive the world. Each agent has a set of sensors, such as fire-location (are any cells within my radius-of-view on fire?) and road-edge (in what direction does the road continue?). 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 for the cognitive component. An example is the reflex of 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; however, 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 exam-

[5]The agent architecture is described in more detail in [2].

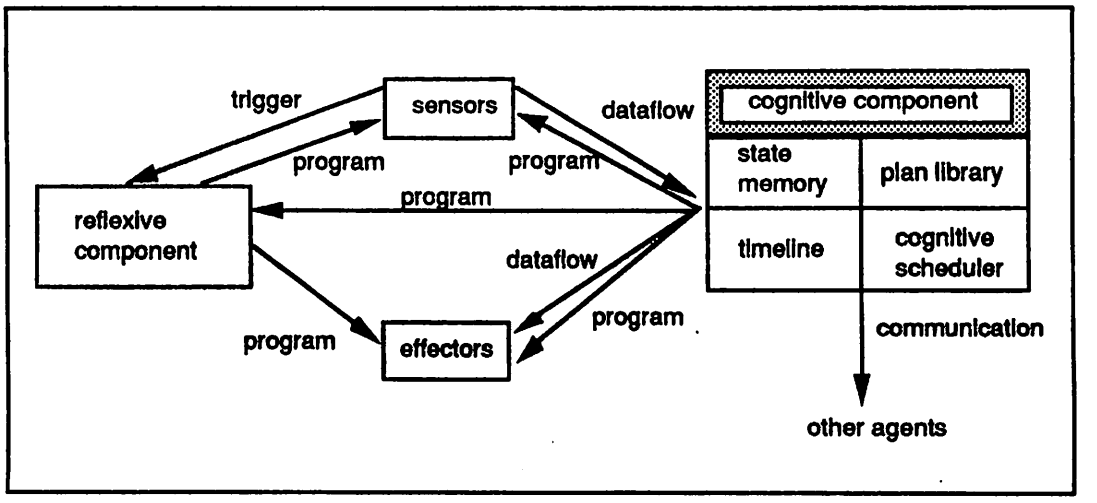

Figure 3: Basic Agent Architecture

ple, 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 movement effector to start moving parallel to the fire. If the cognitive component also programmed the blade effector to put the blade in the down position, the bulldozer would not only maintain a safe distance from the fire, but it would also build fireline 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. Because they respond directly to the environment and so must keep up with it, sensors, effectors, and reflexes operate at the same time scale as the simulation environment

and are synchronized as closely as possible within the discrete event simulator.

The cognitive component receives input from sensors and sends programs to the effectors to interact with the world. It is responsible for data integration, agent coordination, and resource management, in other words, most problem solving activity. This component operates in larger time slices than the others, thus reducing the overhead of context switching, but increasing the possibility of reasoning with outdated information.

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. Actions may be added in 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 resource requirements (e.g., processing time and necessary data). 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, how they do it (the Lisp code for their execution, called the execution methods), and what resources and data, environmental and computational, they require. A plan is a special type of an action. It includes a network of actions related by their data references and temporal constraints.

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. We call this style of planning skeletal refinement with lazy expansion. 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 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 fireboss has far more sophisticated methods for gathering and integrating information than the bulldozer does. It can direct the actions of the bulldozers, while the bulldozers can only make requests of the fireboss. However, the fireboss, unlike the bulldozers, does not know how to get out of the way of the fire because it does not work close to the fire.

Creating a different type of agent requires defining a cognitive component. One can define a set of programmable sensors and effectors (of arbitrary complexity) or use ones that are already defined, and add a set of reflexes to handle situations that require instant response by the agent. To create sensors and effectors, the simulator must be told rates of action under varying environmental conditions, range of perceptions, and other physical capabilities. Creating reflexes involves describing the triggers, the expected output from sensors, and the response, the programming for the effectors. The default cognitive component consists of plans, which are networks of actions available to the agent and tailored to situations in the environment, and methods which describe how to execute the actions. Creating a new cognitive component with the same structure as that described here involves defining a new plan library.

3.2 Structure for Real-Time Control in the Agent Architecture

A number of design decisions in the Phoenix agent architecture 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. The combination of a reflexive and cognitive component accounts for time scale mismatches inherent in an environment that requires micro actions and contemplative processing. Micro actions, such as following a road and keeping out of the immediate range of the fire, involve quick reflexes and little integration of data. Contemplative processing, such as route planning, involves long search times and integration of disparate data such as available roads, terrain conditions, and fire reports. An example of the interaction of the two styles of processing is given in Section 4.1. This horizontal decomposition ensures that the agent can perform reflex actions to keep it from danger and maintain the status quo, while also performing more contemplative actions. This strategy for responding to disparate demands of the environment is advocated by Brooks [1] and Kaelbling [6]; although in both cases, they chose more levels of decomposition for their domains. Our agent architecture, in effect, combines two different planning components: one highly reactive, triggered by specific environmental stimuli and operating at a very small time scale, and the other slower and more contemplative, integrating large amounts of data and concerned with resource management and coordination.

Another design feature that facilitates real-time control is the timeline and its single representation for all actions. Because prospective actions share a uniform representation on the timeline, all problem solving actions have access to the same memory structures and can be monitored and allocated resources using the same mechanisms. All problem solving tasks are subject to the same constraints with respect to resource allocation: how much time is required, what information gathering resources are required, and what data are necessary. This framework allows new cognitive capabilities

to be integrated easily by defining their requirements within the action description language and relying on the timeline and its supportive scheduling mechanisms to temporally arbitrate their allocation.

Structuring plans as skeletons to be filled out at run-time also facilitates real-time control. Plans are only partially elaborated before the agent starts to execute the plan. This deferred commitment exploits recent information about the state of the world to guide action selection and instantiation. Completely deferred commitment, such as in reactive planning [4], is probably not tenable when agents or actions must be coordinated or scarce resources managed[5]. The integration of planning and acting in Phoenix 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 (skeletal plans coordinated on the timeline) that accounts for data, temporal and resource interactions.

4 Integration of Techniques for Real-Time Problem Solving

The simulator exerts variable real-time pressure on agents. How do Phoenix agents respond to real-time pressure? The agent design includes several approaches to managing cognitive resources. One involves tailoring decision components to the appropriate time scale as in the distinction between the reflexes and the cognitive component. Another is through control of processing requirements. This approach enhances the flexibility of actions and the sophistication of control decisions. Providing alternative execution methods for timeline entries ensures a range of choices that vary in their timeliness. Different scheduling strategies for managing the actions on the timeline provide greater responsiveness to real-time constraints. Another approach is an expectation-based monitoring technique that reduces the overhead of monitoring while providing early warning of plan failure. Earlier warning of plan failure affords the planner more time to adjust and more flexibility in possible responses. These approaches are dis-

cussed below.

4.1 Control in the agent architecture

The reflexes and the cognitive component are both responsible for directing the agent's actions. Reflexes take immediate simple actions based on local sensory data and operate at a small time quantum. The cognitive component directs all the effectors over extended periods of time and so operates within longer time slices. As a consequence, the "ultimate" authority in the architecture is the cognitive component. The reflexes direct temporary changes in the settings of sensors and effectors to allow the agent to respond to short time scale demands from the environment, but the cognitive component, at its time scale, may modify the sensors and effectors settings made by the reflexes. Additionally, the cognitive component can program the reflexes, turning them on or off or changing their sensitivity.

Fighting the fire requires the integration of both reflexes and the cognitive component. For example, direct attack of the fire involves a bulldozer building fireline close to the edge of the fire; this plan for containing the fire sacrifices minimal forest area, but necessitates careful attention to the changing fire conditions. A static reflex to avoid fire would preclude working close to the fire. So the cognitive component and the reflexes are coordinated to handle this situation. The cognitive component sets the blade and movement effectors to start the bulldozer building fireline parallel to the fire and the fire edge sensor to attend to the changing edge of the fire. Additionally, it sets the triggering threshold of the avoid fire reflex to be a much shorter distance than normal and turns on a reflex to make minor corrections to the movement effector in response to the changing edge of the fire. It then attends to other cognitive tasks. As the bulldozer builds line, a reflex changes the bulldozer's direction of movement to hug the border of the fire at a respectable distance. If the fire sensor detects that the edge of the fire has gotten too close, the avoid fire reflex will be immediately triggered, will cause the bulldozer to back away from the fire, and will signal the cognitive component of the change. When the cognitive component is next active (see the discussion

of discrete event simulator in Section 2.2 for a description of process activity), it will decide what to do under the changed conditions and may reset the movement effector to continue building fireline or rethink the original plan. This integration assigns the task of immediate self-preservation to the reflexes and that of integrating disparate constraints and ensuring coherent action to the cognitive component.

4.2 Control of processing requirements

Processing requirements can be controlled in two ways: by controlling how much time is used by individual actions and by controlling the overall distribution of time across all actions. Approximate processing [7] and anytime algorithms [3] are methods for controlling how much time is used by individual actions. In these methods, processing time is traded against quality or correctness of solution to satisfy temporal constraints. In Phoenix, algorithms with different processing characteristics, such as those offered by approximate processing and anytime algorithms, are included as *alternative* execution methods. Execution methods, as introduced in Section 3, are lisp code that performs the cognitive actions. Each cognitive action may be executed by one of several execution methods, with differing time requirements and so differing solution expectations. These processing characteristics are represented for each execution method. The Phoenix planner delays the choice of an action's execution method 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.

Alternative execution methods are particularly useful in actions that incur potentially high computation costs with predictable results, such as path planning. Phoenix uses an A* algorithm to calculate paths for bulldozers. It searches the two-dimensional map representation of the world for the shortest travel time path between two points. It expands the current best path incrementally, searching each unobstructed neighboring cell for the best next step. The algorithm is parameterized to work at multiple levels of

resolution, so that search steps could range from 128 meters up to 8 kilometers. A fine resolution search step, 128 meters, yields the shortest path, requiring the least travel time for the bulldozer. However, this resolution requires the most computation (i.e., cognitive resources). The largest search step, 8 kilometers, typically yields a longer path, which requires more travel time, but can be calculated quickly, consuming less computation time. At times it even fails to find a solution, since there are bottlenecks in the map that don't appear at coarse resolution search steps. Each of these resolutions constitutes a different execution method for calculating a path, alternative methods which trade-off cognitive-time for quality of solution. Which of these methods to use requires evaluating the trade-off with respect to judiciously allocating processing time to candidate actions.

The cognitive scheduler controls the overall distribution of cognitive processing time across all actions. At each time step, it selects the next action from the timeline to execute, chooses an execution method for the action, and executes it. Thus, the scheduler is key to controlling the responsiveness of the cognitive component to real-time constraints. The current version of the scheduler for Phoenix is rudimentary and considers only a short horizon for scheduling decisions. It selects the next action for execution based on timeline ordering, action priority and the amount of time an action has been waiting for execution.

Our more sophisticated scheduler (currently being designed) will manage the resource requirements, time and other resources under contention, by constructing an allocation schedule for the timeline. Because the timeline is constructed by lazy expansion, the scheduler also will delay commitment on time allocation by setting specific deadlines and allocations for near term actions and less specific ones for actions further in the future. This strategy favors incremental modifications to the schedule in response to critical conditions. Critical conditions are changes to the world that invalidate the derived schedule, such as an action violating a deadline or unexpectedly contending for resources. The incremental scheduling actions will be similar to those proposed by [9], such as right shifting the schedule, re-ordering actions within a critical window, and

swapping resource allocations between actions.

4.3 Using envelopes to monitor progress

Just as we can explicitly represent the movements of an agent through its physical environment, so can we represent its movement through spaces bounded by failure or other important events. These spaces are called *envelopes*. Typically, one dimension of an envelope is time, and the others are measures of progress. For example, imagine you have one hour to reach a point five miles away, and your maximum speed is 5 mph. If your speed drops below its maximum, for even a moment, you fail. As long as you maintain your maximum speed, you are *within your envelope*. The instant your speed drops below 5 mph, you *lose* or *violate* your envelope. This envelope is *narrow*, because it will not accommodate a range of behavior: any deviation from 5 mph is intolerable. Most problems have wider envelopes. Indeed, real time systems should be designed to ensure that narrow envelopes are the exception, not the rule.

The following problem illustrates a wider envelope. A bulldozer has one hour to travel five miles, as before, but its maximum speed is 10 mph. It starts slowly (perhaps the terrain is worse than expected). After 40 minutes it has traveled just two miles. It can still achieve its goal, but only by traveling at nearly maximum speed. Clearly, if the agent waits 40 minutes to assess its progress, it has waited too long, because an heroic effort will be required to achieve its goal. In Phoenix, agents check their envelopes at regular intervals, hoping to catch problems before they get out of hand. One near-term research goal is to develop a theory of envelopes that will tell us when and how often they should be checked.

Agents check *failure* envelopes, which tell them whether they will absolutely fail to achieve their goals, and *warning* envelopes, which tell them that they are in jeopardy of failure, or succeeding beyond expectations. Typically, there is just one failure envelope but many possible warning envelopes. To continue the previous example, the bulldozer would violate a warning envelope if its average speed drops below 5 mph, because this is the speed it must maintain to achieve its goal. Violating this envelope says, "You can

still achieve your goal, but only by doing better than you have up to this point." These concepts are illustrated in Figure 4. The failure envelope is a line from "30 minutes" to "five miles," since the bulldozer can achieve its goal as long as it has at least 30 minutes to travel five miles. The average speed warning envelope is a line from the origin to the goal, but the bulldozer violated that envelope immediately by traveling at an average speed of 3 mph. In fact, it moved perilously close to its failure envelope. The box in the upper right of Figure 4, defined by the current actual progress, illustrates that the agent can construct another envelope from any point in its progress. In this example, the new failure envelope is extremely narrow: the bulldozer must cover the remaining 3 miles in 20 minutes, maintaining a 9 mph average speed that is close to its top speed.

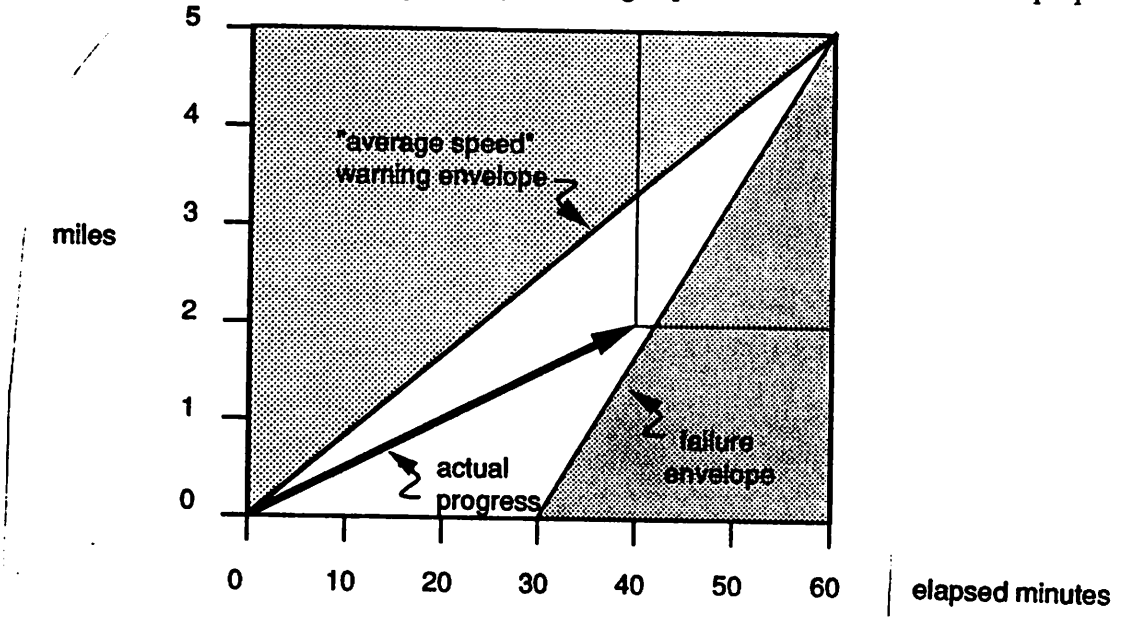

Figure 4: Depicting actual and projected progress with respect to envelopes

Plan Envelopes and Agent Envelopes. We distinguish between the envelopes of individual agents and those of multi-agent plans. In Phoenix, plan envelopes are maintained by the fireboss agent, who coordinates subordinate agents such as bulldozers[6]. Because the environment changes, global plans may be put in jeopardy even if agents are

[6]The hierarchical organization of coordination and communication among Phoenix agents is described in [2].

making progress that, from their local perspective, is well within their envelopes. Figures 5a and 5b illustrate plan envelopes as they are currently implemented in Phoenix. Figure 5a shows the current state of the fire, its projected boundaries after one and two hours, and the firelines that three bulldozers are expected to cut. By projecting where the fire will be, then adding some slack time, the fireboss anticipates that the last of these lines will be cut an hour before the fire reaches it. It creates two envelopes based on this anticipation: a failure envelope that will be violated if the slack time falls under an hour, and a warning envelope violated if the slack time rises above three hours. Thus, if the fire grows faster than projected, or if the bulldozers fail to build line as quickly as expected, the plan will fail. On the other hand, if the slack time increases above three hours, the fireboss will be warned that there is too much slack time, which indicates that the fireline is being built further from the fire than is necessary. Thus, the envelope is parameterized to *expect* between one and three hours of slack time at any point during plan execution. Should the slack time increase dramatically, the warning envelope will be violated, suggesting that the fireboss consider resource conservation measures such as redirecting the bulldozers to build line closer to the fire or sending some back to base; should it decrease below one hour, the failure envelope will be violated, triggering reassessment of the current plan with the possibility of replanning.

this figure will be split into two parts and reworked slightly... I've hardwired references to Figures 5a and 5b for now...

In Figure 5b, we see the actual progress of the fire. After one hour it has grown less than expected and falls short of the one hour projection line. The amount of slack time grows correspondingly, since the fire will take longer to reach the fireline if it continues to grow at this slower rate. The change in slack time is graphed at the bottom of Figure 5b, which shows the plan to be maintaining its expected progress. During the next hour, however, the fire grows more rapidly than expected - so rapidly, in fact, that the failure envelope is violated. Sometime during this interval the fireboss will check the plan envelopes and discover the violation. A typical response to this violation is to send one or more additional bulldozers to increase the rate at which the fireline is dug.

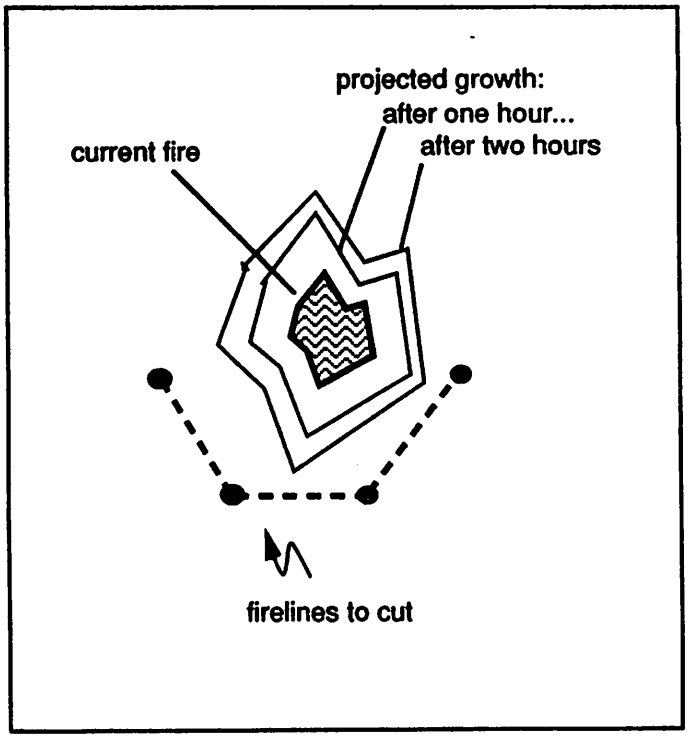

Figure 5: A growing fire with projections of fire spread and fireline used to create plan envelopes

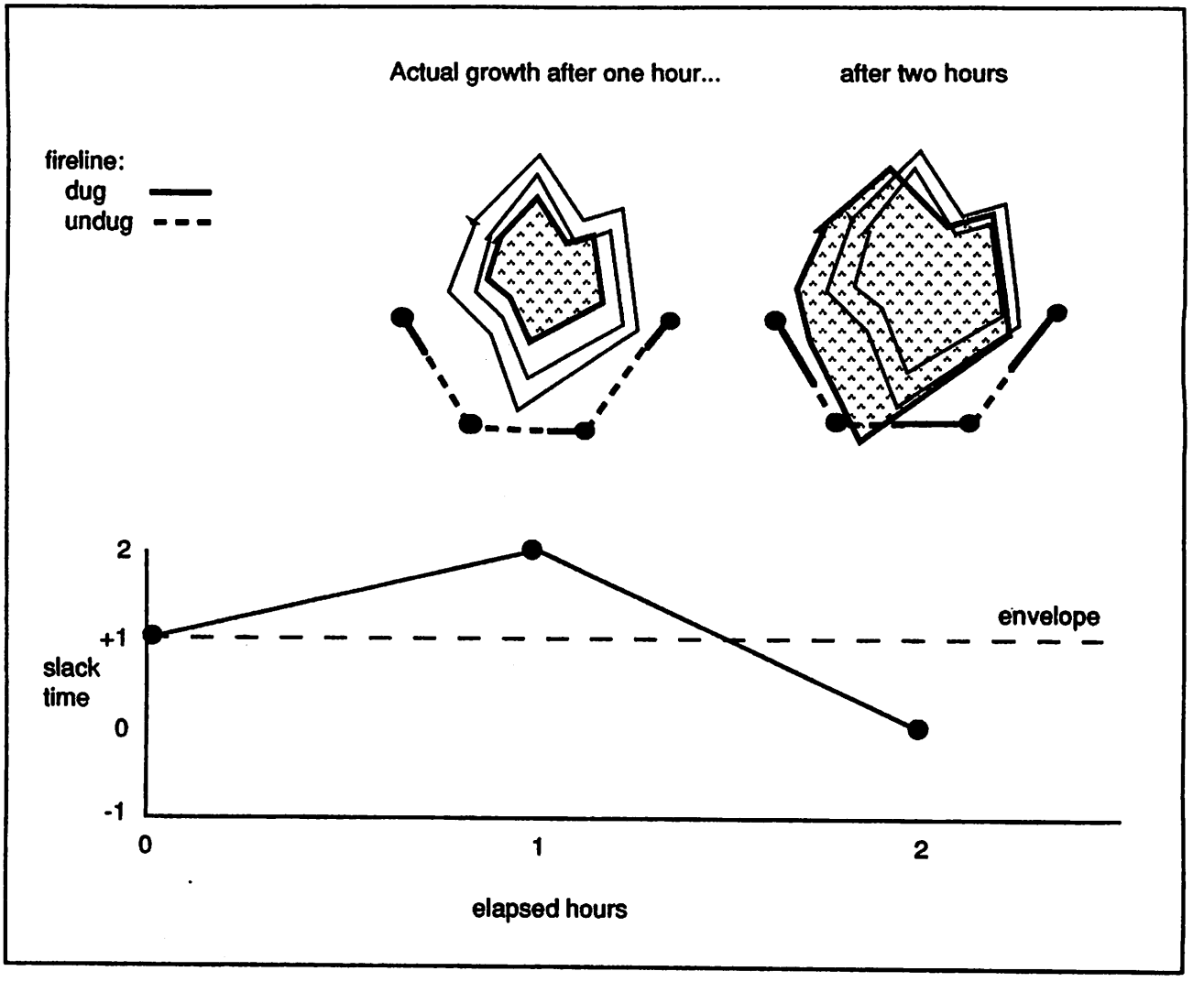

Figure 6: A plan envelope for maintaining slack time

Local plans for single-agent activities are carried out by field agents under the direction of the fireboss. These plans are monitored by the field agents using agent envelopes. In the example above, bulldozers are instructed by the fireboss to proceed to points on the projected fireline and dig assigned segments. Each is given an expected finish time for its segment, and creates an envelope to monitor its progress with respect to this finish time. If it cannot maintain its envelope, it reports a violation to the fireboss. The fireboss assesses the ramifications of this failure for the overall plan (by consulting the plan envelope, among other things), and takes corrective action if necessary. None of the bulldozers reports an envelope violation in the preceding example; from each bulldozer's local perspective, progress is satisfactory. In the absence of failure reports, the fireboss assumes the fireline will be finished in the projected time frame. The plan fails due to a change in the environment that violates the global expectations of the fireboss about the growth of the fire.

The Utility of Envelopes. A planner can represent the progress of its plan by transitions within the plan's envelopes. Progress, failures and potential failures are clearly seen from one's position with respect to envelopes, whereas this information is not always apparent from one's position in the environment.

Envelopes function as early warning devices in two ways. First, explicit warning envelopes alert the planner to developing problems. Second, failure envelopes can tell an agent it has failed long before its allocated time has elapsed. In Figure 4, for example, the agent knows it has failed as soon as it crosses the envelope. A third kind of early warning has yet to be implemented: Just as a planner can project the course of events in its environment, so it can project its progress within its envelope and, particularly, when an envelope might be violated. A simple projection method is extrapolation. For example, if we checked the envelope in Figure 5b, after 75 minutes we would see a "downward" trend. By linear extrapolation we could estimate when the envelope would be violated. Of course, the downward trend may reverse or level out, but sometimes it will be worthwhile to have the projected time of envelope violation

despite its uncertainty.

Envelopes integrate agents at different levels of a command hierarchy: A fireboss agent formulates a goal and corresponding envelope parameters, and gives them to a subordinate agent with the following instructions: "Here is the goal I want you to achieve. I don't care how you do it, and I don't want to hear from you unless you achieve the goal or violate your envelope." The subordinate agent then works independently, not monitored by the fireboss. In the case of a bulldozer, it figures out where to go, how to avoid obstacles, and how to keep clear of the fire, until its goal is achieved or its envelope violated. Meanwhile, the fireboss is free to think about other agents, other goals, or to replan if necessary. Envelopes grant subordinate agents a kind of autonomy, and grant superordinate agents the opportunity to ignore their subordinates until envelopes are violated.

We have yet to develop cognitive scheduling mechanisms to take full advantage of envelopes. The design of these mechanisms is motivated by the following questions: How often should envelopes be checked? Should we adopt a fixed interval or a dynamic one, and if the latter, what execution methods will determine when to check next? When should agents project envelope violations and how should they use the projections? Given that checking a plan envelope, or projecting progress with respect to it, may involve collecting and integrating information from the environment and all the participating agents, the cognitive overhead of these activities can be considerable and must be carefully scheduled.

4.4 Conclusion

To date, most of the work in the Phoenix project has been devoted to the design and implementation of the simulator and the agent architecture. Our research currently focuses on defining methods for controlling the architecture and expanding the plan library for specific agents.

The methods described in Section 4 are the building blocks of an integrated approach to addressing the real-time demands of the Phoenix environment. Execution

methods will be selected and allocated processing time based on estimates of other processing requirements by the cognitive scheduler. The cognitive scheduler will use envelopes to monitor the processing requirements of the potential actions and the execution of pending actions. For example, ideally the fireboss should give a bulldozer some instructions and an envelope, and then forget about the agent until the envelope is violated or the agent succeeds. Imagine that the fireboss gave its agent a warning envelope, and eventually the agent reports that it is violated. The fireboss can now assess the agent's progress within its envelope. By projection it can determine when the agent is likely to achieve its goal. How far is the agent from its goal? If it is nearby, the delay might be acceptable. But if the agent still has a long way to go, then it will violate its failure envelope relatively soon. By determining that the agent's current action is probably doomed to failure, the fireboss can start formulating an alternative goal immediately, and can know when it should redirect the agent, assuming progress doesn't improve. Currently, envelopes provide the data necessary for this kind of reasoning, but our cognitive scheduling algorithms are not sophisticated enough to use it. As both envelopes and the cognitive scheduler are developed, the two will become integrated toward more informed scheduling, and an agent better able to respond to the real-time pressures of the fire fighting domain.

References

[1] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[2] Paul R. Cohen, Michael Greenberg, David Hart, and Adele Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *to appear in AI Magazine*, Fall 1989. also Technical Report, COINS Dept, University of Massachusetts.

[3] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Minneapolis, Minnesota, 1988.

[4] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, Washington, 1987.

[5] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 677–682, Seattle, Washington, 1987.

[6] Leslie Pack Kaelbling. An architecture for intelligent reactive systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, pages 411–424, 1987.

[7] Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[8] National Wildfire Coordinating Group, Boise, Idaho. *NWCG Fireline Handbook*, November 1985.

[9] Peng Si Ow, Stephen F. Smith, and Alfred Thiriez. Reactive plan revision. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages

77–82, Minneapolis, Minnesota, 1988.

[10] N. S. Sridharan and R. T. Dodhiawala. Real-time problem solving: Preliminary thoughts. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems*, Detroit, Michigan, 1988.

[11] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, October 1988.