RADC-TR-89-259, Vol V (of twelve)
Interim Report
October 1989

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Distributed Planning for Dynamic Environments in the Presence of Time Constraints

Syracuse University

Robert A. Meyer, Susan Conry, Paul R. Cohen, Victor R. Lesser

This effort was funded partially by the Laboratory Director's fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-259, Vol V (of twelve) has been reviewed and is approved for publication.

APPROVED: Northrup Fowler III

NORTHRUP FOWLER III
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, Jr.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-259, Vol V (of twelve)
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC)	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)	
6c. ADDRESS (City, State, and ZIP Code) Science & Technology Center, Rm 2-296 111 College Place, Syracuse University Syracuse NY 13244-4100	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b. OFFICE SYMBOL (If applicable) COES
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008	
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO
62702F	5581	27	13

11. TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 Distributed Planning for Dynamic Environments in the Presence of Time Constraints

12. PERSONAL AUTHOR(S)
Robert A. Meyer, Susan Conry, Paul R. Cohen, Victor R. Lesser

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Jan 88 TO Dec 88	October 1989	32

16. SUPPLEMENTARY NOTATION This effort was funded partially by the Laboratory Directors' Fund. This effort was performed as a subcontract by Clarkson University and the University of Massachusetts to Syracuse University, Office of Sponsored Programs.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Distributed Planning; Plan Recognition; Artificial Intelligence; Temporal Reasoning
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the fourth year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photointerpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

The specific topic for this volume is the real-time simulation of a distributed planning simulation in the context of a dynamic environment.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Northrup Fowler, III	(315) 330-7794	RADC (COES)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Item 10. SOURCE OF FUNDING NUMBERS (Continued)

Program Element Number	Project Number	Task Number	Work Unit Number
62702F	5581	27	23
61102F	2304	J5	01
61102F	2304	J5	15
33126F	2155	02	10
61101F	LDFP	27	01

NAIC

Northeast Artificial Intelligence Consortium

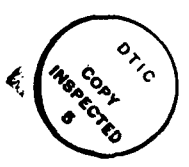
1988 Annual Report

Volume 5

Distributed Planning for Dynamic Environments in the Presence of Time Constraints

Susan E. Conry and Robert A. Meyer
Electrical and Computer Engineering Department
Clarkson University
Potsdam, New York 13676

Paul R. Cohen and Victor R. Lesser
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003



Contents

5.1 Executive Summary

This task is one which started in August of 1988. Since there were only two months' activity during FY88, the bulk of the effort in FY88 has been devoted to delineating the strategy to be employed in building the experimental testbed facility and in beginning the design of the facility.

Our primary goal during the first few months' activity on this task has been one of developing a distributed firefighting simulator. This is necessary in order to provide an environment in which agents can cooperatively plan to contain fires. In order to accomplish this task, a number of issues related to timing, agent synchronization, management of "thinking time" and "acting time", and agent capabilities must be resolved. During FY88, these issues were investigated and the design of a distributed simulator for the firefighting domain was formulated.

The testbed simulator has not been designed simply as a distribution of the existing centralized firefighting simulator. We found that the issues of time and agent synchronization in a multi-agent environment necessitated a complete redesign of the simulation. The new design permits multiple agents to work simultaneously and independently. It includes a facility for defining the characteristics of communications among agents, with available communication media independently specified. In addition, the new simulator will reflect much more realistic terrain representation and a significantly improved fire model than the existing centralized simulator.

Other work on this task this year has been concentrated on formulation of an appropriate agent model and mechanisms for handling time in general and reasoning strategies for adjudicating allocation of time among various cognitive activities in the planner. It seems clear that time is a critical resource for these types of problems. When time is viewed as a resource, proper allocation of time among subtasks is critical in achieving reasonable performance. One problem central to development of heuristics for determining time allocations is that of formulating ways of handling the fact that time can be viewed in more than one way. It seems evident that the CPU time associated with the planner is measured in seconds to minutes, whereas the time associated with acquiring information regarding the state of the fire may be measured in minutes or hours. Reasoning about actions in an environment such as this requires that the planner understand and be able to deal with these extreme differences in scale. Effectively, there are two types of time: "execution time" and "action time" or "internal time" and "external time". These two types of time share some attributes, but are fundamentally different in others (as far as the planner is concerned). Preliminary research concerning a model of distributed planning, agent characteristics, and effective ways of modeling time has been initiated in FY88 and

mechanisms for experimenting with time (as perceived by agents in the system) have been incorporated in the simulator design.

5.2 Introduction

As has been mentioned, we utilize the firefighting domain as an example of a real time planning problem in which time is a valuable resource that must be carefully allocated. It seems evident that in any real time planning environment, it is imperative that planning and situation assessment be handled in an appropriate manner. One approach to the problem involves a system architecture that incorporates a planning model that interleaves planning and monitoring tasks. The planner cannot achieve reasonable performance levels without relying on a situation assessment "agent" to provide timely information. Likewise, the situation assessment task is driven, in part, by the planner. Computation time must be shared among the agents in an appropriate manner.

In the firefighting domain, time can be viewed from two perspectives. Computation time is utilized by the cognitive agents in determining what actions should be taken to further their goals. Execution time is required to accomplish various tasks in pursuit of those goals. Time management becomes a problem of managing both computation time and execution time in such a way that each module is allocated time "proportional" to its current ability to further the firefighting effort.

In the preliminary work that has been done this year, our attention has been focused on development of a distributed firefighting simulator, on a preliminary design for an agent, and on development of strategies for sharing computational resources among cognitive agents in a dynamically changing environment. The work discussed in this document should be viewed as a reflection of preliminary design efforts. It is not yet mature. As this work does mature, it is anticipated that many of these ideas may undergo significant revision.

In the sections that follow, we discuss our distributed firefighting system's structure, a preliminary design for the cognitive component of an agent, and various aspects of managing time in a real time environment.

5.3 Distributed Firefighting System Structure

There are four parts to the system (now called Phoenix).

- The multi-tasking environment (which controls scheduling of all the tasks). (ie. to burn the fires, move bulldozers etc.)
- The fire simulation, which burns the fire . (The fire simulation is a task.)

- The agents. The environment allows for any number of agents. In theory, an agent can be implemented with an arbitrary lisp program. In practice, we restrict all agents to use a common top-level control loop (the agent model).
- The user interface

In the paragraphs which follow, we give brief descriptions of each of Phoenix's major components.

5.3.1 The multi-tasking environment

The environment provides for the scheduling and synchronization of all tasks. Tasks fall into one of two types (based on how time is measured): simulation-tasks and agent-tasks. Simulation-tasks are responsible for measuring time on their own. As an example, when the fire-simulator is executed, it must return to the environment how much time has elapsed. If the fire is updated in 5 minute increments, and the simulator is executed at time T0, the simulator returns to the environment 'restart me at time T0+5'. Elapsed time for agent-tasks is a function of CPU usage. Each agent has a ratio of CPU time to real-world-time. Thus elapsed time for an agent is CPU-usage*(real-time/CPU-time). This allows us to vary the amount of "computes" available to an agent with respect to the changing world. Different agents can be "quicker" then others. It is important to have both types of timing. With it, we can vary the speed of the agents, holding the speed of the environment constant. Also, from an agent's point of view, time has meaning in this model. We can say "the fire moves at 5 miles/hour" and expect that to be true regardless of how many agents there are or what the ratios are. The multi-tasking environment keeps processes synchronized to within about 5 minutes (real-world-time). (For the time being, 1-CPU second = 5 minutes, as it is not possible, on the Explorer, to interrupt processes more often than once per CPU second.)

5.3.2 The fire-simulation

The fire-simulation is a detailed forest-fire simulator. When calculating the rate of spread of the fire, the simulator currently takes into account: wind-speed, slope, temperature, humidity, fuel moisture content, type of ground cover (hard wood, softwood, agriculture, meadow, chaparral) and surface features (roads, rivers, fire breaks and buildings). The fire model also contains time of day, season and cloud cover, but these are not yet implemented. The model provides for 17 types of ground cover, though we only use the five listed above. The effects of wind-speed and slope are non-linear. Given all of these features, the fires form interesting shapes, which are not easily predictable by an agent.

5.3.3 Agents

Agents in a distributed planning environment must be able to interleave planning, execution, monitoring, and communication in a reasonable manner. The firefighting simulator provides facilities for specifying available communication media. This allows incorporation of a reasonable communication model. The cognitive component of an agent is largely concerned with planning, and a basic planning paradigm utilizing skeletal refinement is currently envisioned. Since planning and monitoring must be interleaved (to enable response to changing conditions), agents must also be able to monitor the current context. Design of agents is an ongoing task.

5.3.4 User interface

The user interface allows friendly access to Phoenix. The interface allows the user to start/stop agents, inspect/modify the scheduler state, start fires, and create agents (such as planners, bulldozers, watch towers). Each agent has its own knowledge of the fire, and it is possible to display each agent's view of the world.

5.4 Preliminary Design for an Agent

Our preliminary design for an agent can be viewed at two levels: the high level architecture of our agent and a more detailed description of the cognitive component. In the initial design stages, we have concentrated on the cognitive component of the agents because elements of the cognitive component are critical to time management issues.

We use the term "cognitive component" to refer to all of those tasks an agent performs that require CPU time. Probably the most encompassing element of the cognitive component is the planner, though monitoring is also critical. The planner is the process that decides what to do next and does it. The planner must decide not only what should be done next in the world (Explorer National Park), but also what it should think about next.[1] Deciding what to do requires determining what it is possible and desirable to do (more generally thought of as planning). In addition, because this is a real-time domain with many time constraints on actions, an agent must be concerned with scheduling actions at particular times. Finally, an action cannot be dismissed after a command is sent; in a rapidly changing world, the action may not produce the expected results and so must be monitored. Based on these constraints, we have formulated the basic architecture/methods as a preliminary design for performing the functions of the planner.

[1] We make the assumption that the agent can only "think about" one thing at a time.

5.4.1 The Planning Loop

In general, planning proceeds by a cycle: create plan, execute plan and monitor plan. The problem with viewing planning as endless repetition of this cycle is that it does not fit the exigencies of planning in real-time domains and it assumes that generating a single plan is the only cognitive activity. Consequently, it does not permit partial instantiation interleaved with execution and monitoring. For these reasons, we propose an alternative view that treats each of the these three functions, along with others, as tasks that can be expanded into specific methods and must be scheduled just as actions in the outside world must be scheduled.

5.4.2 Tasks for Planning

As mentioned above, "create plan", "execute plan" and "monitor plan" are three steps in an abstract skeletal framework for planning. At the least, skeletal plans must contain a context of applicability and a set of actions. Additionally, the constraints of the domain (in particular, time constraints) require that the skeletal plans include information necessary to ordering and scheduling the actions and estimating their resource requirements. Thus, the basic appearance of skeletal plans could be:

Key: $(cond\_A \wedge cond\_B \wedge cond\_C) \vee (cond\_D \wedge cond\_E \wedge cond\_F)$

Feasibility conditions: *extension of the key matching that requires more complexity or calculation, should be used only after the first round of filtering*

Resource requirements: type of resources and amount required

Real-time requirements: Three factors are of importance relative to real time constraints:

- CPU time required to execute the feasibility conditions
- CPU time required to instantiate this plan
- time required to execute the plan

Plan steps: *corresponds to what the planner needs to do to execute a plan*

The key refers to the contents of "working memory" and corresponds to internal or environmental conditions. The form of the keys will more or less correspond to OPS5 literals. Feasibility conditions permit two levels of filtering: a quick memory based filtering and a more complex filtering. The most obvious use for the second level of filtering is in estimating probable future states. For example, if the focus fire is likely to spread quickly sometime soon (as over grassland), it would be beneficial to pick a

plan that accounts for that, but doing so may require searching some distance from the fire.) Resource requirements may preclude using a plan (for example, the plan may require 1000 gallons of fuel and only 600 are available). Real-time requirements are needed both to decide whether this plan is appropriate and to generate a timeline and schedule actions.

The plan steps, in their current form, include the actions that need to be performed and the temporal constraints on their execution. The actions in the plan steps are those performed by the planner, in other words cognitive actions. Actions performed in the outside world are achieved by sending messages to effectors. By having cognitive actions represented in plans, we can reason about how to allocate cognitive time and coordinate external and internal constraints. An example of a cognitive plan is:

1. Allocate two bulldozers, B1 and B2
2. Calculate goto point, P1
3. Send message to bulldozer B1 "Goto P1"
4. Send message to bulldozer B2 "Goto P1"
5. At-same-time
   - monitor
   - In-sequence
     (a) Calculate left goto point, P2
     (b) Send message to bulldozer B1 "Goto P2"
   - In-sequence
     (a) Calculate right goto point, P3
     (b) Send message to bulldozer B2 "Goto P3"
6. Calculate rendezvous point, P4
7. At-same-time
   - monitor
   - Send message to bulldozer B1 "Goto P4"
   - Send message to bulldozer B2 "Goto P4"
8. Acknowledge rendezvous at point P4

These actions may be directly executable or may need to be expanded. The temporal constraints relate actions that must be executed sequentially and those which can be executed in parallel.

Given a tentative representation for skeletal plans, we can sketch the rough planning loop as follows:

1. lookup appropriate plans
2. select subset of plans to be executed
3. instantiate plans
4. map actions onto timeline
5. execute actions
6. monitor progress

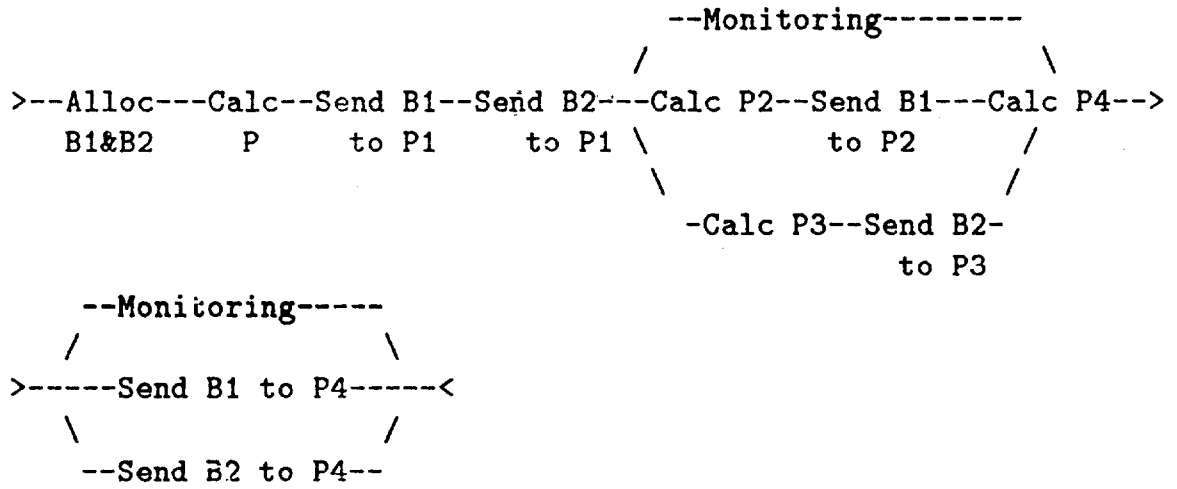
5.4.2.1 Lookup appropriate plans As implied by the skeletal plan contents, plans are indexed by their context. Looking up plans requires matching the contents of global memory and local context of the matching task (e.g., goal and plan in progress) to the key in the skeletal plans. This stage will return only complete matches and will retrieve all possible plans that match. As part of the retrieval, plan variables that refer to the context will be filled in with values; for example, if a plan includes a reference to the wind direction, then the exact value of the wind direction will be filled in during lookup.

5.4.2.2 Select subset of plans to be executed Though the lookup matches the environmental context to the key, the plans retrieved may still be undesirable. This step filters plans based on heuristic estimates of resource requirements (for example, compute time, real time, cost, fuel, and people), feasibility conditions, and possibly syntactic considerations (for example, number of preconditions, specificity of match, and coverage.) If after filtering there are still too many plans to consider, then a subset must be chosen to work on. A subset is selected because the members of the subset may share initial steps and so not require an early commitment until later environmental considerations force one. These plans may also refer to different aspects of the context (for example, multiple fires.)

The knowledge for selecting plans should reside in both the filtering algorithms and in the skeletal plans. The knowledge in the filtering algorithms is manipulable (able to be tailored to present needs) in the sense that many filtering methods may be available and the one selected clearly influences the filtering decision. How the knowledge should reside in the skeletal plans will depend on the nature of the filtering algorithms we design.

5.4.2.3 **Instantiate plans** Instantiation of plans requires replacement of plan variables with values. It is intended to be a relatively simple operation. The skeletal plans will include constraints on instanstiation. As was the case with plan selection, several methods will be available for plan instantiation.

5.4.2.4 **Map actions onto timeline** The plan steps as described previously can be mapped almost directly into a timeline. The timeline is used by the scheduler to determine what action should be done when. When mapped onto the timeline, the plan mentioned earlier looks like:



Note that the actions on the timeline are exactly those in the original plan; they have not been expanded yet. We adopt a strategy of least commitment expansion so that actions are expanded only when necessary. At the latest, actions will be expanded as they get scheduled. However, actions may be expanded earlier in response to explicit requests for expansion.

5.4.2.5 **Execute actions** This step depends on the nature of the actions. Most of the actions will need to be further expanded. Even some of the actions that appear to be primitive (for example, send message) may actually expand to *create envelope* followed by the execution of the action.

Assuming the actions are primitive and need not be further expanded, the actions should be directly executable by the cognitive component. If the action involves sending a message to an effector, then the message should, of course, be in a form the effector can either directly execute (in the case of "brainless" effectors) or expandable (for more autonomous effector agents.)

5.4.2.6 Monitor progress For the most part, this action is yet to be specified. However, we have determined that monitoring progress will usually refer to maintaining envelopes on actions. Monitoring will be scheduled as other actions are except that they may repeat (execute) at regular intervals or at specific times.

5.4.3 Role of Monitoring and Envelopes

There are multiple levels of monitoring that must occur. For example, the planner must watch the world for new or unexpected events, such as the initial fire-sighting, or the sighting of a second fire. It must monitor each plan in progress to measure its effectiveness and coordinate plan steps. Agents must also be monitored to see whether each of their actions has its intended effect. In the following paragraphs, we briefly outline our early thoughts concerning the types of monitoring that must be handled and how this can be accomplished.

5.4.3.1 Monitoring It is clear that the current *context* (global state of the simulation) needs to be checked at intervals to notice changes in the state of the simulation world. This could be a simple matter of checking the same context data structure referenced by the skeletal plan matcher. Basic events such as the fire characterization, the rate of spread of the fire, and the wind speed and direction must be monitored.

In addition, each plan must be monitored to be sure that the plan steps are completed in the proper order and have their intended effect, and that deadlines are met. This monitoring must be done during each phase of the plan, since it is used to determine whether the plan is in danger of failing. It is recursive in nature, since plans can be subplans of other plans. Thus monitoring the inner and outer plan is necessary to be sure that each is progressing satisfactorily. In addition, provision must be made for multiple plans to be executed (and monitored) at the same level, since several plans could be running concurrently. The result is that there can be multiple plan level monitoring activities on the timeline during any given interval, each watching the activities of a single plan.

Details that are tracked in these plan monitors include the original context information the plan matched, such as the fire characterization. These can be monitored by simple predicates (is the wind direction still x?) and functions (if the wind direction has changed, how does the fire characterization change?). There are more complicated relationships that must be monitored at the plan level, such as the coordination of external events in the plan. Some sense of the overall progress of the plan must also be maintained.

At another level, each plan step must be monitored to measure its progress and effectiveness. Many plan steps involve coordinating external events and maintaining joint deadlines among different agents and activities. For example, if two bulldozers

are sent to a rendezvous point, monitoring is required to ensure that the point is still realistic - if the fire encroaches on the point faster than anticipated, it might be impossible to serve as the rendezvous point.

Finally, actions themselves must be monitored, including external actions carried out by agents such as bulldozers. In our first pass we expect to have the planner monitor these actions, but eventually the agents will be responsible for monitoring themselves. Agents must be monitored for the timeliness and effectiveness of their actions. Bulldozers must keep to their given time deadlines to coordinate with other agents. Their success at linebuilding can also be monitored - they can build line slowly and carefully when the situation allows, or they can speed up (leaving gaps) when in a hurry.

5.4.3.2 Envelopes We expect that envelopes will be used for these monitoring activities. When a fire is detected, a world level envelope must be created for it representing its characterization and rate of spread. When a plan is instantiated, a corresponding envelope must be created that contains the predicates and functions needed to insure that the expectations the plan is based on remain as predicted. Many of these expectations are represented as dependencies on the world level envelope around the fire. These are maintained outside the plan, since several plans could be executing simultaneously to fight one fire. In addition to the overall plan envelope, individual envelopes for plan steps must be created when those steps are started, and must be deleted or marked as done when the plan steps finish. Thus the plan must coordinate the creation and deletion of envelopes to correspond to the initiation and completion of those plan steps. Similarly, individual agent's actions are monitored using envelopes that are created and deleted as these actions are initiated (communicated to the agent) and completed.

The knowledge needed to create and maintain envelopes is stored in various places. Plans might be required to set up world level envelopes, as in the plan that executes when a fire is first noticed. Plan envelopes are stored with the skeletal plan, as are the envelopes for steps within the plan. Envelopes for atomic actions are stored with the knowledge about those actions.

World level envelopes are set up as needed to monitor the world (possibly as a by-product of executing a plan). A plan level envelope is set up when the plan is instantiated. As the plan is followed, individual plan step and atomic step envelopes are created and deleted as those steps are instantiated (within the cycle of interleaved plan-step instantiation and execution).

Much of the information needed for envelopes and scheduling depends on the results of previous steps. Activities such as the mental calculation of a path for a bulldozer give expected travel times for the bulldozer, which are used to create an envelope and to set up deadlines on the timeline for mental activities associated with

the succeeding bulldozer action. These dependencies must be stored with the skeletal plans.

The scheduling of envelope creation and maintenance requires decomposing plan steps into smaller parts. A coordinated plan step to send two bulldozers to a rendezvous point requires the creation of three envelopes - one for each bulldozer and one to coordinate their arrival. The cognitive step of creating each envelope must be scheduled on the timeline, and monitoring time must be allotted to each one.

5.5 The Timeline

It is clear from the previous discussion that the timeline represents the set of actions that must be performed by the cognitive component and the temporal constraints that exist between those actions. The timeline includes all of the cognitive actions that may be executed, with the obvious exception of unusual conditions whose occurrence cannot be predicted. Consequently, the timeline presented earlier is actually only a fragment of the system's timeline. The entire timeline would include other plans that are also active, global monitoring actions, and perhaps monitoring actions for entire plans.

5.5.1 Top Level Interpreter

Because we are reasoning both about what to do in the world and what to think about internally, we end up planning on two levels. Most of the discussion in the previous sections has pertained to planning and monitoring in the external environment, retrieving and instantiating firefighting plans. However, referring to those actions with respect to their cognitive effects and mapping the cognitive actions onto a timeline simplifies the relationship between the two levels.

The actions mapped onto the timeline provide general guidelines for what should be done by the cognitive component. We still need to interpret and reason about these guidelines. The basic top-level control loop should have the form shown in Figure 1.

At the most abstract level, the cognitive component operates based on a scheduling strategy. The scheduling strategy biases the scheduling decisions and perhaps the methods used to implement basic planning tasks (e.g., selecting and instantiating plans.) For example, when the planner is running far behind schedule, a *panic mode* strategy might be appropriate so that only the most important actions get scheduled and only the quickest methods for doing them are selected. There may be a high start-up cost associated with any change in scheduling strategies. The scheduling algorithms often require particular data structures to maintain scheduling information

```
Loop
    Select scheduling strategy
    Loop
        Startup scheduling paradigm
        Loop
            timeline-update := next(timeline)
            Incremental timeline update
            Monitor internal envelope
        until too many changes to do incremental update
    until method is no longer appropriate
endloop
```

Figure 1: Top Level Control Loop

and gathering that information may require searching the entire timeline. Consequently, it is not likely that scheduling strategies will be switched frequently. (Some of the specifics of potential scheduling strategies will be discussed in the next section.)

In the innermost loop, the scheduling algorithm determines what action to perform next and executes it. Some of the possible actions may produce side effects that concern the contents of the timeline. These side effects must be accounted for in the timeline, both by updating the contents of the timeline and also by updating the data structures maintained by the scheduling algorithms. If we assume that the large majority of these changes will be minor variations or refinements on finish time, each scheduling algorithm should provide for incremental updating of its own internal structures. Finally, in the innermost loop, internal (cognitive) envelopes will be updated.

Each cognitive plan should have its own working memory for keeping track of progress and internal variables.

5.5.2 Scheduling the Timeline

At any given time, the timeline contains all the cognitive actions the agent would like to perform. In addition, the timeline contains precedence constraints between the mental actions and deadlines for some actions. The scheduling problem is this: *Given the timeline and information about the real-time nature of each mental action, decide which mental action to perform next, and how much time to allocate to that action.* At any given time, there is a current scheduling strategy in effect. This strategy is selected within the top level loop of the cognitive interpreter. Methods for

choosing which action to do next are provided with the strategy. The function F will be used to denote the function that decides which action to do next and the amount of time to allocate to that action. (F is defined by the current scheduling strategy).

One scheduling strategy may be "panic-mode." When in panic-mode, all decisions, including scheduling decisions, must be made as quickly as possible. So F simply returns the first action it finds that can be executed, and allocates the minimum time necessary to that action. A more typical strategy may be "minimum-slack-time-first" (MSTF). MSTF calculates the slack time available for each action in the timeline. F returns the action with the smallest slack time. Slack time can be allocated to actions based on some priority scheme, or saved for the future when there may be a higher demand for computes.

It is clear that to do this, we need a set of scheduling strategies and method for selecting a strategy. The following paragraphs describe one approach to scheduling, the information the scheduler has available at run-time to make scheduling decisions, and several potential scheduling strategies.

In order to completely specify a scheduling strategy, each of the following parameters must be clearly defined:

context applicability When is this strategy appropriate? For example, "lots to do in a short time."

initialization-function This is a function to initialize the scheduler's global state. For example, with an MSTF strategy, a initial pass through the timeline must be done to calculate slack times.

F Given the timeline and the scheduler's global state, calculate which action to do next and how much to allocate to it.

update-function This function processes the changes to the timeline made by the selected actions. In the case of MSTF, the function would update slack times.

monitor? This describes how to decide if it is time to try another scheduling strategy.

The scheduler has several sources and types of information available to assist in the decision making process. Among these are:

Precedence constraints The timeline contains information about the order in which mental actions must be done.

Deadlines The timeline contains information about desired start and end times for mental actions. Since the deadlines may be soft, a more detailed description of the deadline must (may) be given. Is the deadline strict (start/end at a specific

time) or relative (start/end after/before a time)? How hard is the constraint? Is it OK to be a little late?

Durations Associated with each action (or method for performing) is a set of durations.

Priority How important is each action? This may be used to allocate slack time in MSTF.

Atomic Preemptable Some actions can be preempted and resumed later.

Local scheduling preferences Actions may have specific scheduling preferences. For instance, some actions would like to be done as late as possible.

5.5.2.1 Some scheduling strategies

Panic-mode Panic-mode scheduling is applicable when there is mental overload. In this situation, we want to spend as little time thinking about scheduling as possible. Panic-mode allocates as little time as possible to the task with the highest priority. (Of course, this may be exactly the wrong strategy to apply. It may be necessary to do careful reasoning about the timeline to decide exactly which tasks can be safely delayed and which ones cannot. On the other hand, we do not want the system thrashing in the scheduler)

Minimum Slack First Minimum slack time first is based on one of the set of scheduling algorithms that come from a traditional PERT/CPM analysis. When using MSTF, the scheduler does a forward and backwards pass through the timeline to calculate the earliest start time (EST) and latest start time (LST) for each mental activity. By definition, slack time (ST) can be calculated by $ST = EST - LST - duration$. When ST for any task is negative, we know it is impossible to come up with a schedule that meets all its deadlines. Assuming the schedule is feasible (positive ST for all tasks), this scheduling strategy executes the action with the minimum slack time first. The amount of time allocated to the task is a function of the time required by the task and the amount of slack time in the system. Since tasks with small ST are more sensitive to the uncertainties in the "mental time domain", they are executed first.

Problems with this approach: (1) When the knowledge about deadlines is bad (uncertain), measures of ST don't mean much. (2) There is no obvious definition of slack time when activities can't be done in parallel. ST depends on the order that actions are executed. (3) What do we do when the schedule is infeasible? Negative values of slack time tell us how late an action will be.

Earliest Latest Finish Time First ELSTF is another PERT/CPM algorithm. In this case we calculate the latest time each action can conceivably be finished and execute the action whose deadline is earliest first. (Theoretically speaking, when there is a feasible schedule, ELSTF can generate schedules that violate deadlines whereas MSTF always guarantees feasibility). This approach is subject to the same problems as MSTF. ELSTF requires about half as much time to execute as does MSTF.

5.5.3 An Alternative Approach

An alternative approach to scheduling the timeline involves a hybrid scheduling strategy that blends symbolic reasoning about time allocation with more traditional scheduling mechanisms. Such an approach employs flexible decision making and control strategies that should not incur the high degree of overhead associated with changes of scheduling strategy that has been mentioned previously.

This approach to scheduling assumes (as do the other strategies that have been discussed) that actions on the timeline have estimates of computation time and execution time associated with them as well as an assessment of their criticality. This strategy differs from others in that each action type also has a set of heuristic rules associated with it. These rules are used to assess the degree to which it is advisable to schedule an action. They reflect qualitative factors associated with each action, as opposed to such metrics as slack time, which are quantitative. When triggered, an assessment rule places a positive or negative endorsement on some action being considered by the scheduler.

The fundamental scheduling strategy can be summarized as follows:

1. Endorse actions on the timeline with the aid of heuristic assessment rules.
2. Perform a cost/benefit analysis based on strength of endorsement, criticality of action, computation time required, and execution time required.
3. Rank actions based on the cost/benefit analysis.
4. Schedule on the basis of the ranking, consistent with the partial order of the timeline.

There are some obvious difficulties with a strategy such as this. The most significant of these difficulties is that there may not be sufficient time available to the scheduler to perform thorough analysis relative to deriving endorsements. To obviate some of these difficulties, the use of progressive reasoning is incorporated.

If there is an imminent emergency, the scheduler has no time to "think" and all analysis relative to endorsement is bypassed. Scheduling is performed based only on

the assessment of criticality associated with each action (and, of course, the partial order inherent in the timeline).

For cases in which there is time for the scheduler to think, we have identified three levels of reasoning that appear to be relevant. At the first level, all rules that can be applied without requesting additional (or more recent) information or performing computations are triggered, and a first level assessment is made. On the second level, a "request list" is formed. This request list reflects information that would help the scheduler arrive at a more informed (hence better) schedule. This level is concerned with updating old information and/or verifying parameters on the request list. A (more informed) assessment is made at the end of this level of reasoning. Finally, in the third level, parameters perceived to be inaccurate or less accurate than desired are recomputed, and a final assessment is made. If the scheduler's time to think runs out during a particular level of reasoning, the ranking determined at the previous level is returned.

Progressive reasoning applied in this context should embody the reactive-reflective spectrum of decision-making employed by the scheduler. It is interesting to note that the planning and monitoring tasks become interwoven in the second and third levels of reasoning. For example, projections about the extent of the fire in a certain area, or updates of the front of the fire, may be requested by the scheduler during these levels, as the scheduler may need this data to make a more informed decision. The problem we see is to define this interaction rather than attempting to avoid it.

5.5.4 Viewing Time as a Resource

In order to gain perspective as to how time can be allocated using this hybrid type of strategy, we view time as a resource, and cast the problem in terms of a resource allocation problem. In this section we identify three different types of "time resources", characterize their properties, and describe interactions among these three types. We then mention a number of constraints arising in this problem, several sources of contention for resources, and finally, discuss some heuristics for allocating time.

For our purposes we define three types of time: act-time, think-time, and idle-time. Loosely speaking, act-time corresponds to the time it takes to perform a "real world" action, think-time is computation time, and idle-time is characterized as the absence of either think-time or act-time (depending on the subtype). We further assume that there are three "reasoning agents", the scheduler (S), the planner (P) and the monitoring agent (M). The system also has some number of "effector agents," agent*1*, ..., agent*k* for some *k*. These effector agents represent the agents in the field performing actions to control the fire. Examples include bulldozers, crew, helicopters, planes, etc. The "reasoning agents" consume think-time and idle-time; the "effector agents" consume act-time and idle-time. (For the time being, we assume that effectors

are "brainless".) With respect to time allocation, the hybrid scheduling strategy has the following goals: to maximize useful "effector agent" actions, taking advantage of as much parallelism as possible, to maximize the usefulness and efficiency of CPU usage, and to minimize all idle-time.

Suppose that we are given an interval of time, (t, T). This gives rise to a "universal pool" of time resources available, for some value of t and T. This pool (call it R) can be viewed in two different ways. In one sense, R can be viewed as an interval divided lengthwise into two bands, one labeled "think-time" and one labeled "act-time." The act-time band is further divided lengthwise into k bands, each smaller band labeled "agent$i$" where i varies from 1 to k. When all think- and act- time has been allocated, these bands will have been divided into resources, much as an interval can be divided into subintervals. In agenti's act-time band, this corresponds to the time agent$i$ takes to complete any actions it performs within (t, T), as well as the position within (t, T) in which they were performed. Any "gaps" correspond to idle-time. Initially, before any resources are allocated, the bands are undivided, so the resources themselves are not yet defined.

This brings us to the second way in which R can be viewed. We may consider R as a collection of sets of possible resources to allocate, only one element of which is ultimately realized when the available time interval has been allocated. The problem of optimally allocating time (when it is viewed in this manner) is clearly intractable. To see this, consider an example. It is evident that act-time and think-time are measured in different units. For sake of illustration, suppose act-time is measured in 2-second intervals, and think-time in 1-second intervals. Suppose the total length of the interval (t, T) is 6 seconds. Then think-time may be divided among the reasoning components as follows: 1(S) + 1(M) + 1(P) + 3(idle), which means that the scheduler has 1 second of CPU time, then the monitor, then the planner, followed by a three second interval when the CPU is idle. The same 6 second interval could also have been allocated as 1(M) + 2(S) + 3(idle), or as 1(S) + 1(M) + 1(P) + 1(M) +2(P). In order to consider all possibilities, one must consider all partitions of 6, along with all permutations allowing for different orderings of usage, as partially illustrated above. Using the same example scenario, agent1's act-time could be divided up as follows: action1 takes 4 seconds, action2 takes 2 seconds; or agent1 is idle for 2 seconds, action1 takes 4 seconds, etc. To get one set of possibilities in R we take one particular allocation for think-time, together with one allocation for each effector agent's act-time. All possible such sets together comprise R. With this view of time, the process of allocating time can be viewed as one of progressively eliminating sets from R, with the goal of having an "optimal" set remain as the single set finally left in R. Heuristics that reduce the remaining alternatives in R very quickly are necessary.

In the following sections, we summarize various observations that we have made concerning properties of time (when viewed as a resource to be allocated among

various tasks). We anticipate that these observations will result in further refinement of our model of time and in development of heuristics for effective time allocation.

5.5.4.1 Characterizations of Time Resources Time, when viewed as a resource, has a number of significant attributes. Some of these attributes are common to all "types" of time, while others are specific to one kind of time. Various attributes of importance in time allocation are mentioned in this section.

Act-time

1. consumed by "effector agents"
2. allocated by the scheduler
3. a specific instance of allocated act-time has 3 properties:
   (a) length of duration
   (b) position within a time interval (t, T)
   (c) "effector agent" associated with it
4. for a given "effector agent" an instance of act-time, once consumed, cannot be re-used
5. within any given subinterval of (t, T), there may be several instances of act-time currently being consumed (by different agents)
6. measured in units of r(act), probably given in units of minutes to hours
7. can be consumed concurrently with think-time (there are some restrictions, which will be noted later)
8. purpose is to effect an action; at the end of an instance of act-time, the agent using it has performed an action

Think-time

1. consumed by "reasoning agents", the scheduler, monitor, and planner
2. allocated by the scheduler
3. a specific instance of allocated think-time has 2 properties:
   (a) length of duration
   (b) position in (t, T)

4. a specific instance of think-time, once consumed, cannot be re-used

5. within any given subinterval of (t, T), there may be sequential but not concurrent instances of think-time being consumed

6. measured in units of r(think), probably given in units of seconds to minutes

7. can be consumed concurrently with act-time

8. purpose of think-time is decision-making and computation

Idle-time

1. there are 3 types of idle-time:

   (a) (think) CPU is not in use

   (b) (act) a particular agent is not acting

   (c) (act) no agent is acting

2. idle-time is not allocated specifically, but is defined as the absence of either think-time or act-time being consumed

3. a specific instance of idle-time has 2 properties, the precise characteristics of which are determined by the allocation of think-time and act-time:

   (a) length of duration

   (b) position within (t, T)

4. idle-time of the "act variety" has an "effector agent" associated with it

5. measured in units of either r(act) or r(think), as appropriate

5.5.4.2 Interactions Between Types of Time

1. All act-time instances must be preceded by some think-time instances. Thus, an act-time instance must be preceded by some scheduler think-time and possibly some planner or monitor think-time specific to that action.

2. Some think-time instances may be preceded by an act-time instance. This could happen, for example, if the scheduler needed more information for decision-making. (This constitutes an example of the interweaving of monitoring and scheduling mentioned above).

3. Some act-time may be allocated within scheduling.

4. Idle-time's precise characteristics are determined by the those of the surrounding think-time or act-time.

5. Think-time and act-time may be consumed concurrently, when thinking is relative to implementing future actions.

Since the ratio r(act)/r(think) is large, there are two consequences of (2) and (3) above. First, act-time should not be allocated within scheduling unless the total amount of time allocated to the scheduler is large, and can be measured in terms of r(act). Secondly, should this kind of act-time be allocated, the scheduler would send a request for information and could not continue on its current line of thought until that information was provided. Thus there is the potential for a large piece of idle-time defined in the think-time band. This could result in a significant loss of usable computation time. Measures to avoid this must be devised.

5.5.4.3 Contention for Time Resources Time is a scarce resource in real time environments. Agents compete for computation time, as do tasks within an agent. This section summarizes a number of observations concerning the nature of contention for time.

Act-time

1. Inter-agent
   (a) no contention for act-time between different "effector agents" for independent, non-mutually exclusive actions
   (b) mutually exclusive actions: only one of the agents may be allocated act-time
   (c) if partial ordering of actions known in advance: allocated act-time position property must behave accordingly
2. Within an agent
   (a) mutually exclusive actions: effect of allocated act-time must be considered in advance, and act-time allocated with that in mind

Think-time

1. Contending for think-time:
   (a) scheduler for decision-making
   (b) planner for determining details of implementing an action

(c) monitor for determining details of implementing an action

(d) monitor for computations, such as projections of fire, etc.

(e) planner and monitor for replanning, enable a new decision-making cycle

(f) updating a database describing the state of the fire

2. None of the above can be done concurrently.

3. High level of contention for think-time resources. All activities mentioned in (1) are necessary.

Idle-time

1. no contention; nobody wants idle-time!

5.5.4.4 Constraints on Time Allocation Many of the constraints on allocating different types of time resources have already been noted. In particular, interactions between instances of time give rise to constraints, as do certain aspects of contention among resources. To some extent, these constraints may already be reflected in the timeline.

As already noted, an act-time instance must be preceded by its corresponding think-time, as well as scheduler decision-making time. If there is a deadline by which the action must be completed, this constrains:

- amount of scheduler decision-making time
- allocation of think-time relative to the action
- allocation of think- and act- time relative to other actions

If there is no such constraint on an action, the "beginning point" of its act-time is constrained to be after the "end point" of its think-time (in "wall time").

In addition to the difference in scaling, think-time and act-time have another fundamental difference that can be seen as a kind of constraint. Once an instance of think-time is allocated, there is no potential for extending the total length of that instance. For example, if the scheduler is allocated 5 seconds of think-time, beginning at a particular point in "wall time," then after those 5 seconds are consumed, control of the CPU is granted to the next "reasoning agent." However, if bulldozer1 is given 10 minutes to get from PointA to PointB, and it takes 12 minutes to do so, in some cases this is fine, and bulldozer1 is allowed the full 12 minutes to complete the action. This means that there is less control over an "effector agent" consuming precisely the resource it is given than with a "reasoning agent."

5.5.4.5 Criteria for Allocating Time as a Resource Heuristics are clearly needed in allocating time well. An approach to scheduling that explicitly considers time as a resource attempts to blend qualitative reasoning with more traditional scheduling heuristics. In this section, we mention some of the parameters that affect these heuristics relative to act-tme and think-time.

Act-time Broadly speaking, there are two types of actions considered in this type of system: one that brings about an "effect" in the "real world" and one that increases knowledge. In addition, actions may have originally been placed on the timeline and they may reflect scheduler requests for information.

Factors affecting time allocation for actions placed on the timeline:

- Contextual reasons for wanting the action to be performed:

  Does the situation in question merit this action, and how strongly? Are there reasons that the action is warranted, considering what we know (or think we know) about the future? These kinds of concerns are captured in the rules of our system, and through the use of endorsements.

- Other factors to consider:

  Is there a time constraint on when action must be completed? If not, will the CPU be tied up too long before this action is given a chance? How much think-time is involved in this action? These concerns are incorporated in specific efforts to rank actions, and to consider time itself as a cost in the cost/benefits analysis.

Factors associated with allocating act-time during scheduling:

- Length of allotment of act-time not "too large" (determine upper bound).
- Can the concurrent think-time be used constructively? Don't want CPU to be idle for long periods of time.
- If a request for information is issued by the scheduler and n time units are allotted, the scheduler will wait no longer than n time units for that result. In this case, the scheduler is more likely to want to allocate this time if it is reasonably certain it will get a result within the time allotted.

Think-time

We have already determined that during each scheduling cycle, the scheduler needs at least some minimal time to think, if only to decide to sidestep the full-blown process. Similarly, the global database should be updated on a regular basis. The planner and monitoring functions need some amount of time for replanning purposes,

so the amount of time given to these activities must be determined. Not all actions currently listed on the timeline will necessarily be allocated CPU time. The position in "wall time" and length of an instance of think-time will, to some extent, be decided by the amount of time requested. For example, an action that takes very little CPU time would tend to have higher priority than something that is very computation intensive.

5.6 Status and Concluding Remarks

The distributed firefighting simulator mentioned in Section 5.3 has been implemented. Its user interface is sufficiently flexible as to permit reasonable instrumentation of the system. We have concentrated much of our effort on developing preliminary concepts related to our model of distributed planning and our design of an agent. The issue of effective time management has received particular attention. During the coming year, we intend to refine many of the ideas mentioned in this report and build a rudimentary distributed planner.