# Planning Alternatives for an Intelligent Scheduler

**Terrence Harvey** and **Keith Decker**
Computer and Information Sciences
University of Delaware
Newark, DE 19711
{harvey,decker}@cis.udel.edu

## Abstract

Manufacturing environments are notoriously dynamic. Carefully designed production plans can be ruined by machine downtime, personnel changes, or even traffic between you and your supplier. Automated planning can be very time consuming, and re-planning to meet contingencies may require time that isn't available.

The DECAF agent architecture views the functions of "planning to achieve objectives" and "scheduling start times of specific actions" as separate but interrelated agent components that may operate at different time scales. DECAF has a scheduler that selects actions at runtime based on a dynamic user-specified utility function that considers action quality, cost, and duration. To make good use of the intelligent scheduler, a planner must provide the scheduler with appropriate choices.

Every autonomous DECAF agent also includes an HTN planner that designs plans to make use of the scheduler's intelligence. Previous contingency planners have focused on making a plan responsive to unmet preconditions or the value of certain runtime variables. We explore two new kinds of contingency planning: planning to allow the scheduler to make runtime choices about "pre-planned" alternative actions; and planning alternatives to maximize plan flexibility when the scheduler responds to a changing user-provided utility function.

## 1 Introduction

One of the core, definitional aspects of an intelligent software agent is its ability to flexibly, pro-actively adapt its activities to achieve its goals [Wooldridge and Jennings, 1995]. AI planning techniques are one way to achieve this goal achievement flexibility. On the other hand, flexible reactive adaptation is also a core aspect of intelligent software agents [Wooldridge and Jennings, 1995], but reactivity is often at odds with carefully pre-planned behavior. The development of soft real-time AI scheduling techniques allows for such situational flexibility. We view planning and scheduling as a continuum from deciding "how" to do something to deciding "when" and "in what order". It makes sense to develop software agents with separate planning and scheduling components both because of the different algorithms available, and to explicitly balance the deliberative reasoning an agent does with respect to action choice and reconsideration in highly dynamic, volatile environments.

As soon as one comes to this realization, however, it becomes clear that the ideal planner for a software agent that has an intelligent scheduling component is different from those planners designed to work directly with an execution component. Such a planner must plan not only to achieve goals, but to also provide run-time flexibility in the resulting plan. This includes well-understood contingencies such as runtime variables, precondition failures, and interleaved planning and execution. It also includes some new kinds of contingencies: "design-to-time" planning to achieve deadlines, and "design-to-criteria" planning to react to dynamically changing utility functions.

## 2 Planning vs. Scheduling in DECAF

For DECAF, the traditional notion of BDI "intentions" as a representation of a currently chosen course of action is partitioned into three deliberative reasoning levels: planning, scheduling, and execution monitoring. This is done for the same reasons given by Rao [Rao and Georgeff, 1995]—that of balancing reconsideration of activities in a dynamic, real-time environment with taking action [Russell and Wefald, 1991]. Rather than taking the formal BDI model literally, we develop the deliberative components based on the practical work on robotics models, where the three tier models have proven extremely useful [Bonasso *et al.*, 1997]: here, Planning, Scheduling, and Execution Monitoring. Each level has a much tighter focus, and can react more quickly to external environment dynamics than the level above it. Most authors make practical arguments for this architectural construction, as opposed to the philosophical underpinnings of BDI, although roboticists often point out the multiple feedback mechanisms in natural nervous systems.

The DECAF Planner places actions on a Task Queue for consideration by the Scheduler. In a very simplistic Scheduler, the order might be first-come-first-served, resulting in some very long wait times for tasks. On the other hand, determining an optimal schedule for a set of tasks in a fixed period of time can be an NP-hard problem. The notion of an

*Anytime Agent* [Sandholm and Lesser, 1994] is used to allow agents to search until a specific criteria or quality of answer is achieved. Real-time performance may also be achieved by conditional scheduling of agents using sampling [Greenwald and Dean, 1998]: an initial schedule is laid out and then may be changed based on the sampling results. A third method for achieving real-time results is known as *Design to Time* [Garvey and Lesser, 1991], where the scheduler determines the exact method to achieve an optimal result in the time given. The most recent work has resulted in the development of a sophisticated *design-to-criteria* action scheduler [Wagner *et al.*, 1997] that efficiently reasons about action duration, cost, result quality, and other utility function characteristic tradeoffs. It is for this class of dynamic schedulers that a flexible planner is necessary.

## 3 The DECAF Planner

DECAF (Distributed, Environment-Centered Agent Framework) is a toolkit which allows a well-defined software engineering approach to building multi-agent systems. Each autonomous DECAF agent has its own HTN planner. In order to enter into meaningful negotiations with other agents, each agent needs to be able to commit to achieve certain objectives[1]. In DECAF, we also want agents to be able to commit to high level objectives without necessarily committing to a specific plan to achieve those objectives. For example, a contractor agent should be able to commit to the objective of building a house for a certain price, and by a certain time, without having to specify where materials were going to be purchased, which subcontractors were going to be hired, and the number of nails to be used. Making commitments to high level objectives allows the agent to achieve the objectives in a manner than is most suitable given the environment at runtime. If a subcontractor breaks a leg, the contractor agent can hire a new subcontractor without having to (necessarily) renegotiate and re-commit.

Input to the Planner is a set of predicate objectives joined by a *quality accumulation function* such as AND or OR[2]. A plan consists of tasks and actions arranged to achieve the top level objectives, such as

```
(AND
 (ACHIEVE (MAKE WIDGETS 100 (COST 0 500)(GLOSS HIGH)))
 (ACHIEVE (MAKE WIDGETS 2000 (COST 0 400)(GLOSS MEDIUM))))
```

Actions are leaf nodes in a plan that contain executable code. Tasks in DECAF are really task decompositions, and contain no executable code, but designate how information flows in the plan. Both actions and tasks are written in STRIPS style, with preconditions and effects. The planner matches the effect of an action or task against the goal it is trying to achieve. Predicates are used to describe the preconditions and effects, and can be constraints or goals depending on whether or not they are associated with a knowledge or achievement predicate. For example,

```
(KNOW ?symbol (SYMBOL AMAZON ?symbol))
```

describes a planning goal; but

```
(SYMBOL AMAZON AMZN)
```

has no performative, and so the planner interprets this as a constraint that must hold before a task's effects can be instantiated.

Action and task specifications contain syntactic predicate descriptions of the parameters and provisions they accept. Predicates can be arbitrarily named and used without formal definition, or can be defined in Java class files to have special executable functions (as in [Wilkins, 1990]). It is the unification of such predicates in the descriptions of outcomes and provisions that allows the planner to plan not only task decomposition, but information flow as well.

## 4 Planning to Achieve Objectives

The requirement that agents be able to commit to objectives, not specific plans to achieve them, motivated three features of the planner. First, task decompositions are written in terms of predicate goals. Second, the planner explicitly represents the goals being addressed at each point in the plan. And third, the planner stores the entire task network of each plan.

Having task decompositions written in terms of predicate goals is an important feature of the planning system. Each task separates a particular goal or set of goals into more specific goals, that are in turn further decomposed or achieved by an action. The system plans to achieve the goals at each point in the plan, without an *a priori* preference for a particular way of achieving them.

Planning to achieve predicate goals is useful in a system where many actions in an agent's Plan Library have the same effect. This approach enables the planner to compare all available means of achieving a task, such as comparing the expected utility of an action that achieves a complex goal to the expected utility of a task that decomposes the goal further. Agents can add to their action/task libraries when other agents advertise services that are competitive with existing actions in the agent. Decomposing to specific action names would make this difficult, since existing decompositions in the agent would have to be modified. Under the plan-to-objectives approach, the specification of actions/tasks in terms of goal predicates facilitates additions to the Plan Library, since no modifications to existing decompositions are necessary. This kind of facilitation is important since the DECAF architecture is *open* and agents can enter or leave the system at any time.

The second feature of the planner that supports agent commitment to objectives is the explicit representation of goals at each step in the plan. Because the goals are explicit, the agent knows at every plan step exactly what its *intentions* are at that point. It can then use this information to reason about its commitments and make decisions about how much more plan expansion is necessary before committing to an intention [Rao and Georgeff, 1995; Jennings, 1993].

Third, saving the entire task network of each plan, and the explicitly represented goals therein, is also necessary for an agent to be able to reason over time about committing to a plan or portions of a plan. This requirement results from possible plan failure or any negotiation of a commitment [Jennings, 1993; Decker and Lesser, 1995]. If the planner back-

[1] See [Cohen and Levesque, 1990].

[2] For a description of QAFs in DECAF see [Decker, 1995]

tracks from a failure point, it is critical to know at which points in the plan the agent has committed to an intention to achieve a goal. Furthermore, having explicit goals stored in the plan allows the planner to re-plan without starting from the top objective (this also requires a stored representation of expected changes to state, which is also maintained along branches of the plan).

In the case of negotiating or re-negotiating commitments about a plan, the agent must be able to examine the intentions (explicit goals) of the plan at each step *even after the plan has been expanded*. This requirement was noted by [Young *et al.*, 1994] for systems that had to reason about dialogue. In dialogue, as in negotiating about commitment, sometimes it is necessary to reason not merely about *what* has been said/done, but *why* it was said or done. In this case, having stored action/task names would only record the "what"; recording the explicit goals means that the agent has access to the "why" as well.

Being able to commit to high-level objectives does not imply an inability to commit to a certain plan. Since the goals can be very fine-grained, writing task or action specific goals for a given domain or problem enables the agent to commit to a fully specified plan, though we feel that this undermines a purpose for which DECAF was intended, namely achieving objectives in a dynamic environment.

## 5 Planning in a Dynamic Environment

One way to handle planning in a dynamic environment is to interleave planning and execution [Knoblock, 1995]. This involves developing a plan as far as possible, then halting development of the plan and waiting for sensors to instantiate runtime variables that will influence the development of the remainder of the plan. For real time applications, this requires the planner to complete the plan in real time. The DECAF planner supports this method of interleaved planning and execution, but we view planning in real time as a situation to be avoided when possible. Instead, we would like to plan a range of alternative paths that allow the scheduler to make the best of a dynamic situation.

### 5.1 Contingency Planning

A second way to plan for contingencies in a dynamic environment is to enumerate mutually-exclusive possible environments, create a separate path in the plan for each possibility, and then determine at runtime which environment is the current one. This kind of contingency planning includes CNLP [Peot and Smith, 1992], and planners with explicit sensing operations such as SGP [D. Weld and Smith, 1998], Cassandra [Pryor and Collins, 1996], and UWL [Etzioni *et al.*, 1992]. C-BURIDAN [Draper *et al.*, 1994], which does probabilistic planning and can accommodate noisy sensors, also expects actions to have mutually exclusive outcomes. The DECAF architecture supports this method of planning for contingencies, but it is not yet implemented in the planner.

### 5.2 Planning for a Design-to-Criteria Scheduler

A third way to plan for dynamic environments becomes possible in the presence of an intelligent scheduler. The DECAF Scheduler currently reasons about a characteristic accumulation function (CAF) that considers real-valued representations of task cost, quality, and duration. The DECAF agent planner can choose to provide more than one action to address the same goal(s) in a plan, and then let the scheduler make the choice between actions at runtime based on a utility function and the current environment.

Our planner takes as arguments the agent's initial objectives and a utility function. This function may differ from that used by the scheduler, and can consider more parameters than the scheduler's function[3]. When called, the planner first creates the best plan it can based on the utility function, and the parameters to that function that are stored in each task. Currently, we store three real-valued parameters for post-planning use by the scheduler: cost, quality, and duration.

**An arbitrary set of resource parameters**

Here is a simple example to motivate the consideration of an arbitrary set of resources during planning. Consider an agent that controls the production of widgets. A purchaser is interested in an initial batch of widgets to evaluate the product. Price, specifications and completion time are given by the purchaser; the agent must determine how best to use its resources to fill the order.

The factory is currently producing widgets, but the product failure rate is higher than the purchaser's specs. One option would be to produce the widgets under the current factory setup. This option would minimize two costs: the cost of changes or retooling, and the opportunity cost of lost production during downtime. Because price is fixed, profit would be maximized for this transaction. However, failing to meet specs could jeopardize future business. To properly consider this option, the planner must take into account retooling costs, opportunity costs, and some measure of a task's effect on future business.

A second option would be to retool to meet the failure rate specifications, thus incurring the costs mentioned above. This would have the benefit of increasing purchaser satisfaction, but might run a small risk of missing the purchaser's completion date. This risk could be represented as a probability curve and an associated confidence rating. Retooling may also provide new business opportunities for the higher quality goods (or price the widgets out of the market!).

A third option would be to buy a factory that is currently producing widgets of the desired specifications. However, two more attributes are involved in evaluating this task: planning time, since negotiating with other agents and developing such a complex plan might take days or weeks; and lead time before the acquisition plan is actually executed. This option may have a higher risk of missing the completion date.

Evaluating these three simple views of the task requires the planner to consider six attributes: product quality, retooling cost, opportunity cost, effect on future business, risk of missing a deadline, lead time, and the time spent on the planning process itself. In a real world problem, there could be many different attributes for each of these. For example, product

[3] For the planner, the parameters are a vector of arbitrary size.

quality may be divided into failure rate, finish, and size uniformity. For each attribute, the planner must be concerned with absolute values (e.g. maximum cost $<= 100$) as well as user preferences about the relative desirablity of attributes (e.g. is the customer more interested in reducing cost or increasing quality?).

Because of the infinite variety of task attributes that could be useful for a given planning task, it is important that the planner be able to make use of an arbitrary set of attributes. This way, human task designers can write tasks that use the attributes needed for a particular problem or domain *without modifying the planner*. In the case of the DECAF planner, if attribute limits (such as resource restrictions) and user preferences about relative attribute importance are not specified in the planning goal, the missing attribute will be ignored. For example, in the plan above the user could choose to specify only cost, accuracy, execution time, and planning time, causing the planner to ignore consideration of lead time.

The DECAF planner allows the user to specify an arbitrary set of attributes for the planner to use as guidelines when making decisions about the plan. Each attribute will be paired with a specification of minimum and maximum acceptable values. Plan tasks and actions can be programmed with an arbitrary number of such attributes, and the user may use any or all of them when presenting a planning goal to the planner. Having the planner make use of an arbitrary set of attributes makes the planner useful in a vast array of domains, and allows the use of attributes for novel purposes such as placing constraints on the planning process itself.

**A user-specified utility function**

In addition to the hard limits set in design-to-criteria, choices within those limits are decided by having the planner compare options under a utility function. Allowing users to specify a utility function enables the planner to implement user preferences when making planning decisions about resource use. This gives the user the ability to explicitly control the relative importance of criteria *even when the criteria are not approaching their limits*.

The following examples suggest the flexibility this system allows:

ExecTime max = 50
Utility = $Quality - ExecTime$

With these settings, the planner will choose options that have high quality, low execution time, regardless of cost or other considerations, and will not return a plan with execution time$> 50$.

Furthermore, the user can specify *relative* importance through the use of coefficients and exponents:

Cost max = 100
Quality min = 50
ExecTime max = 100
Utility = $Quality^2 + Cost - ExecTime$

While the semantics of "quality squared" may be unclear in most domains, it is clear that the user wants the planner to be more reactive to changes in plan quality than to changes in cost or execution time.

**Planning alternatives**

Once the Planner chooses the best tasks according to the utility function, it adds alternatives to those tasks that can be chosen by the scheduler at runtime. It does so by inserting an OR node in the plan. The alternative task selected by the Planner may be the second best according to the utility function, or the Planner may choose the most dissimilar option, i.e. the option that is *farthest* in unit parameter space from the original choice[4]. The point of choosing a dissimilar (with respect to the parameters) task or action is to provide the scheduler with an alternative that might look good under a very different utility function than the one that led to the first choice.

Planning alternatives for the scheduler provides advantages under several circumstances. Foremost, for our purposes, is that it allows the user to change the scheduler's utility function each time they use the plan. For example, on one run the user can specify a utility function that places the highest priority on result quality, and the result might not arrive for an hour. On the next run, the user can execute the same plan, but specifying a utility function with an emphasis on duration, thus incurring a risk that the result returned may not be the highest quality possible. In each case, the same plan is used, but the scheduler can make a last minute decision based on the user-provided utility function.

A one-dimensional example of planning optional dissimilar tasks is the design-to-time paradigm. By selecting time as the only component of the utility function, and selecting dissimilar branching, the planner will generate plans that vary widely in the amount of time they take to execute. For some kinds of tasks, extra time should result in extra quality, so this kind of planning allows the *scheduler* to maximize quality at runtime by choosing the task that uses the highest percentage of the time available for the task.

The value of spending time in planning to create alternatives increases as a function of the number of times the plan is executed with different user utility functions (assuming that the alternatives improve runtime or reduce re-planning.) We expect some plans that are generated by the DECAF planner to be used many times. Plans that deliver a basket of information on a particular stock may be invoked several times a day, and a user may wish the agent to plan overnight to generate a high quality information package every morning, and then run quickly just before lunch to seek newer information. If the agent is running other plans as well, the scheduler may benefit by having choices in each plan that allow it flexibility.

One example of a situation where scheduler choice is valuable is a plan that includes a long series of actions of uncertain duration. Planners typically know only a programmer's estimate of the time for a given action, or perhaps some history, so this kind of uncertainty is common. If the early part of the plan is being executed faster than expected, a scheduler with choice may be able to select higher "quality", longer duration actions for the remainder of the schedule. On the other hand, if early actions take too long, then the scheduler can choose the shortest remaining actions available in an attempt to meet

[4]The Planner could also be set up to select multiple alternatives; we have yet to fully explore even the simple case of similar vs. dissimilar.

a deadline.

Some other reasons for planning for scheduler choice include

1. creating choices for the scheduler's utility function, either similar to or different from an earlier choice;
2. increasing the likelihood of runtime success by planning alternatives for tasks/actions that have a low confidence of success (based on reasoning about environment or prior success);
3. accommodating or taking advantage of changes in the environment between plan time and runtime (consider finishing a plan before a commitment to facilitate a task arrives from another agent, and then being able to use the facilitation at scheduling time);
4. avoiding time spent backtracking and re-planning during runtime by "re-planning" at plan time;
5. preconditions that may be met at scheduling time, but weren't at plan time;
6. providing as many actions with additive quality characteristics as possible, under a SUM node, to allow the scheduler to "design to time"[Garvey and Lesser, 1991].
7. a decision cannot be made at plan time (e.g. if utility is based on runtime information; currently the utility functions passed to the DECAF scheduler are based on static behavior profiles.)

The DECAF agent planner implements to some degree all but the last reason for planning choices.

## 6 Related Work

Of the planners that plan for mutually exclusive contingencies (including CNLP [Peot and Smith, 1992], SGP [D. Weld and Smith, 1998], Cassandra [Pryor and Collins, 1996], UWL [Etzioni *et al.*, 1992], and C-BURIDAN [Draper *et al.*, 1994]), none are HTN planners. Reasoning and negotiating about commitment to high level intentions requires explicit representation of intentions/goals to be available after the plan has been expanded. This aspect of the DECAF planner is easy to represent in an HTN, but would probably be difficult in other representations.

Full-scale HTN planners have capabilities not shared by the DECAF planner, which must be small enough to work inside an agent within a multi-agent system. SIPE-2, for example, places constraints on variables during planning and uses causal theory to determine the effects of actions [Wilkins, 1990]. In contrast, the DECAF Planner only maintains constraints on plan attributes. The DECAF Planner's critics are also very weak, in part because the tasks of information gathering it performs do not tend to have interactive or harmful side effects.

The planner that is most similar to the DECAF planner is the planner for RETSINA [Paolucci *et al.*, 2000]. Like DECAF, RETSINA keeps the whole plan network for a plan, has a separate scheduler and planner, can interleave planning and execution, and has runtime provisions to allow looping and conditional behavior. RETSINA does not plan to objectives, however. RETSINA decomposes tasks to the names of lower level tasks, while DECAF decomposes to predicate goals. RETSINA uses rationale-based monitors [Veloso *et al.*, 1998] to manage changes in the environment during planning; our current work on the DECAF planner is concerned with managing an environment that changes *after* planning, and before or during scheduling.

## 7 Summary

The design of the representation of actions and information flow in DECAF necessitate an HTN planner that plans to achieve objectives. The DECAF planner is designed to produce flexible plans through its use of scheduler choice, an arbitrary set of resource parameters, and a user-specified utility function. We expect that planning for an intelligent scheduler will increase the usefulness and flexibility of every agent's plans and decrease the overall time spent in planning.

The simple problems the DECAF Planner has addressed so far have suggested that our approach has merit. So far our implementation has focused on the task of information retrieval, but we would like to extend this work to the manufacturing domain, as we feel the strengths of the DECAF planner offer great promise here.

## References


[Bonasso *et al.*, 1997] R.Peter Bonasso, David Kortenkamp, and Troy Whitney. Using a robot control architecture to aoutomate space shuttle operation. In *Proceding of the Ninth Conference on Innovative Appications of AI*, 1997.

[Cohen and Levesque, 1990] P. Cohen and H. Levesque. Persistence, intention and commitment. In P. Cohen, J. Morgan, and M. Pollack, editors, *Intentions and Communication*, pages 33–69. MIT Press, Cambridge, MA, 1990.

[D. Weld and Smith, 1998] C. Anderson D. Weld and D. Smith. Extending graphplan to handle uncertainty & sensing actions. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Madison, WI, July 1998.

[Decker and Lesser, 1995] Keith S. Decker and Victor R. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, San Francisco, June 1995. AAAI Press. Longer version available as UMass CS-TR 94–14.

[Decker, 1995] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995. `http://dis.cs.umass.edu/~decker/thesis.html`.

[Draper *et al.*, 1994] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Intl. Conf. on A.I. Planning Systems*, June 1994.

[Etzioni *et al.*, 1992] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation*

*and Reasoning*, San Francisco, CA, October 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[Garvey and Lesser, 1991] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. COINS Technical Report 91–72, University of Massachusetts, 1991. To appear, IEEE Transactions on Systems, Man and Cybernetics, 1993.

[Greenwald and Dean, 1998] Lloyd Greenwald and Thomas Dean. A conditonal schdeuling approach to designing real-time systems. *The Journal of AAAI*, 1998.

[Jennings, 1993] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[Knoblock, 1995] C. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proc. 15th Int. Joint Conf. on A.I.*, pages 1686–1693, 1995.

[Paolucci *et al.*, 2000] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara. A planning component for retsina agents. In M. Wooldridge and Y. Lesperance, editors, *Lecture Notes in Artificial Intelligence, Intelligent Agents VI*. Springer Verlag, 2000. forthcoming.

[Peot and Smith, 1992] M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proc. 1st Intl. Conf. on A.I. Planning Systems*, pages 189–197, June 1992.

[Pryor and Collins, 1996] L. Pryor and G. Collins. Planning for contingencies: a decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[Rao and Georgeff, 1995] A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, San Francisco, June 1995. AAAI Press.

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, MA, 1991.

[Sandholm and Lesser, 1994] Tuomas Sandholm and V. Lesser. Utility-based termination of anytime agents. CS Technical Report 94–54, Univ. of Massachusetts, 1994.

[Veloso *et al.*, 1998] M.M. Veloso, M.E. Pollack, and M.T. Cox. Rationale-based monitoring for continuous planning in dynamic environments. In *Proc. 4th Intl. Conf. on A.I. Planning Systems*, pages 171–179, Pittsburgh, PA, June 1998.

[Wagner *et al.*, 1997] T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.

[Wilkins, 1990] D. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, 6(4):232–246, November 1990.

[Wooldridge and Jennings, 1995] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[Young *et al.*, 1994] R. Michael Young, Martha E. Pollack, and Johanna D. Moore. Decomposition and causality in partial order planning. In *Proceedings of the Second International Conference on AI and Planning Systems*, Chicago, Illinois, 1994.