# Planning Ahead to Provide Scheduler Choice

Terry Harvey and Keith Decker
Computer and Information Sciences
University of Delaware
Newark DE 19716
{harvey, decker}@cis.udel.edu

## ABSTRACT

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 [31]. 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 [31], 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. This paper includes some example plans generated by an agent using the planner that will help motivate these features.

The paper is organized as follows. First, the DECAF agent architecture is briefly presented, giving context to the planning and scheduling components along with an overview of DECAF's information flow and execution representations. The DECAF Planner is then presented in more depth, including motivation for its features. The potential benefits of planning alternatives for scheduler choice will be described, along with some simple plan-and-execute run times from the system. Last, a brief description of how the DECAF planner differs from other current systems is presented, along with a look at the direction of future research on the planner.

### 1.1 The DECAF Agent Internal Architecture

DECAF (Distributed, Environment-Centered Agent Framework) is a toolkit which allows a well-defined software engineering approach to building multi-agent systems. The toolkit provides a stable platform to design, rapidly develop, and execute intelligent agents to achieve solutions in complex software systems. DECAF provides the necessary architectural services of a large-grained intelligent agent [7, 24]: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis. This is essentially the internal "operating system" of a software agent, to which application programmers have strictly limited access.

DECAF provides an environment that allows the basic building block of agent programming to be an agent action, or a pre-specified subtask (collection of agent actions). These building blocks are then chained together by the DECAF planner. This paradigm differs from most of the well known agent toolkits, which instead use the API approach to agent construction (e.g., [17]). Functionally, DECAF is based on RETSINA [24] and TÆMS [5].

To augment the functions of the agent Planner, the control or programming of DECAF agents can be provided via a

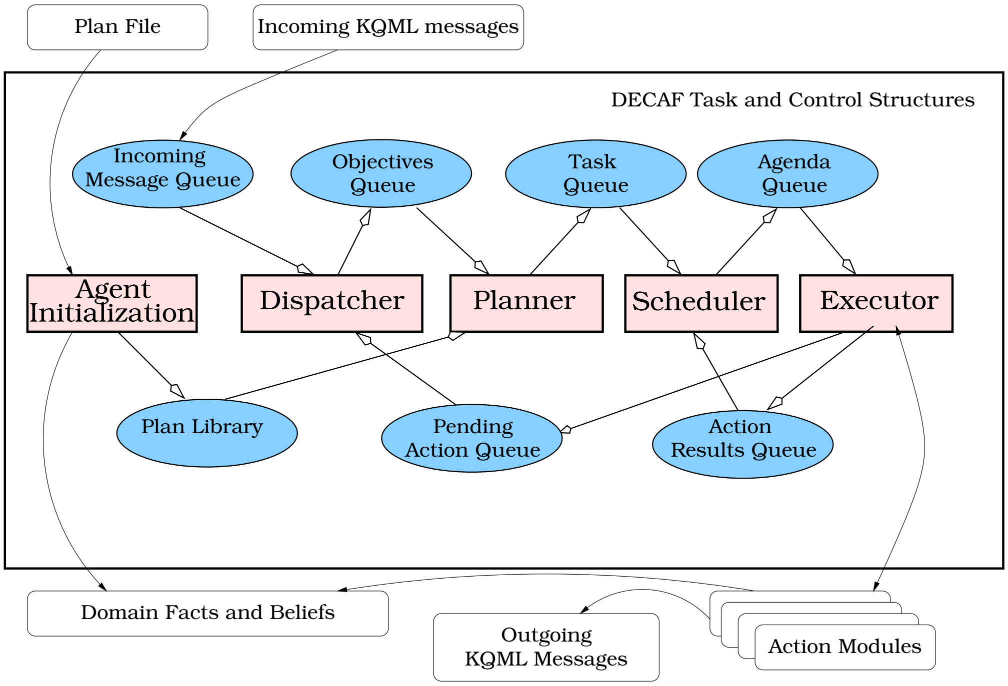


**Figure 1: DECAF Architecture Overview**

GUI called the *Plan-Editor*. The Plan-Editor can be used to construct hand-coded hierarchical subtask networks to be merged into larger, more complete plans by the DECAF Planner, or to view or edit plans generated by an agent using the planner.

Figure 1 represents the high level structure of a single DECAF agent. Structures inside the heavy black line are internal to the agent architecture and the items outside the line are user-written or provided from some other outside source (such as incoming KQML messages).

As shown in Figure 1, there are five internal execution modules (square boxes) in the current implementation, and seven associated data structure queues (rounded boxes). DECAF is multi-threaded, and thus *all modules execute concurrently, and continuously* (except for agent initialization).

The Planner monitors the Objectives Queue and plans for new goals, based on the action and task network specifications stored in the Plan Library. A copy of the instantiated plan, in the form of an HTN corresponding to that goal is placed in the *Task Queue* area, along with a unique identifier and any provisions that were passed to the agent via the incoming message. The Task Queue at any given moment will contain the instantiated plans/task structures (including all actions and subgoals) that should be completed in response to all incoming requests and local maintenance or achievement goals.

## 1.2 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 [19]—that of balancing reconsideration of activities in a dynamic, real-time environment with taking action [21]. 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 [1]: 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.

Once an action from the Task Queue has been selected and scheduled for execution, it is placed on the *Agenda Queue*. 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* [22] 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 [12]: 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* [11], 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 [26] that efficiently reasons about action duration, cost, result quality, and other utility function characteristic trade-offs. It is for this class of dynamic schedulers that a flexible planner is necessary.

## 1.3 Modeling actions and utility

DECAF's underlying Hierarchical Task network (HTN) representation ties together two pieces of work: Williamson's work on information-flow representations resulting in RETSINA

[30, 29], and Decker's work on representations of how local and non-local action executions effect those characteristics over which an agent expresses preferences (via a utility function) resulting in TÆMS [5, 26].

### 1.3.1 RETSINA Information Flow

The unique contribution of the RETSINA information flow representation used in DECAF is the declarative description of the information requirements of actions and the information producing abilities of actions [29]. This is in addition to the traditional precondition and effect representations used in planning systems. The information needs of an action are represented by a set of *provisions*. Provisions can be thought of as a generalization of plan action parameters and runtime variables, in which each provision has an associated *queue* of values. This information may be queued statically at plan-generation time or dynamically during plan execution. An action is *enabled* when there is at least one element queued for each of the actions provisions. Upon execution, the provision is consumed. *Parameters* are a subset of action provisions that are *not* consumed when an action runs (and thus do not involve a queue of values). When an action completes it produces both an *outcome* and a *result*. The outcome is one of a finite set of pre-designated symbols (e.g. the outcomes of CNLP or the observation labels of C-BURIDAN). The result is an arbitrary piece of information. *Provision Links* designate information flow of results from the outcomes of actions to the provisions of other actions.

### 1.3.2 TÆMS

TÆMS task structures are abstraction hierarchies whose leaves are instantiated basic actions or "executable methods". At a basic level this is similar to HTN (Hierarchical Task Network) or TCA (Task Control Architecture) approaches to action representation[9, 23]. Additionally, TÆMS allows the specification of dynamically changing and uncertain task characteristics that effect an agent's preferences (utility) for some state of the world, including tasks with hard or soft deadlines. A TÆMS specification also indicates relationships between local and non-local tasks or resources that effect these agent preference characteristics. Thus it extends HTN ideas toward specifying "worth-oriented" domains [20]. Recent extensions to TÆMS have included named provision relationships and multiple outcome specifications [29, 26]. In utility theory, agents have preferences over possible final states (action or plan outcomes), and preference-relevant features of an outcome are called *attributes*. A substantial body of work exists on relating attribute values to overall utilities [27]. At its core, TÆMS is about specifying these attributes and the processes by which they change—what we call a model of the task environment. In DECAF, we use TÆMS specifications to focus on three common attributes, *quality*, *cost*, and *duration*.

**Actions.** A DECAF *action* represents the smallest unit of analysis. For the purpose of utility calculation, each action has a probabilistic model, called the *behavior profile*, which specifies the likelihood of each outcome, and the probability distribution function for the quality, cost, and duration associated with each outcome.

**Tasks.** A DECAF *task* (or *subtask*) represents a set of related subtasks or actions, joined by a common *quality accumulation function*. For example, in an AND/OR tree, an AND task indicates that all subtasks must be accomplished to accomplish the task, while an OR task indicates that only one subtask needs to be accomplished. Since TÆMS is about worth-oriented environment modeling, it uses continuous rather than logical quality accumulation functions (for example min instead of AND, max instead of OR[1]). For example, subtasks may be joined by a SUM quality accumulation function, indicating that as many subtasks as possible should be attempted. DECAF allows the explicit specification of a *characteristic accumulation function* for each characteristic (e.g. quality, cost, duration).

## 2. THE DECAF PLANNER

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[2]. 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 sub-contractor 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[3]. A plan consists of tasks and actions arranged to achieve the top level objectives, such as (AND (ACHIEVE (UPDATE NEWS IBM)) (KNOW ?value (PRICE IBM ?value))). 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.

## 2.1 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.

[1] The full set of quality accumulation functions, including alternate definitions for AND and OR, is discussed in [4].
[2] See [2].
[3] For a description of QAFs in DECAF see [4]

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 [19, 13].

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 [13, 6]. If the planner backtracks 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 [32] 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.

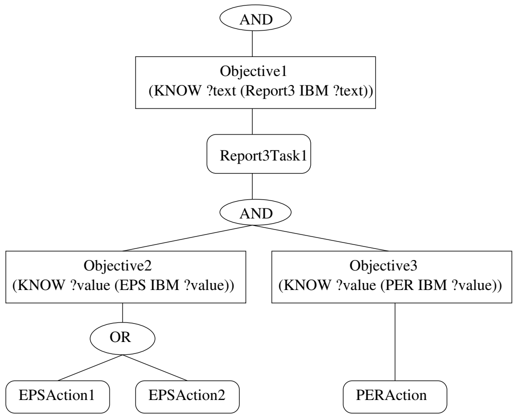


**Figure 2: A simple plan generated with the dissimilar branching option turned on, so the second action was selected for its "distance" from the first, without regard to the utility function (see Figure 4). Only decompositions are shown, not data flow.**

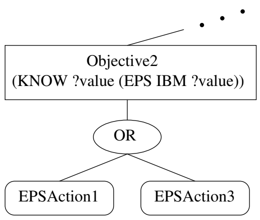


**Figure 3: The left branch of the same plan, but with similar branching turned on. The second action was also chosen to fit the utility function. See Figure 4.**

## 2.2 Planning in a Dynamic Environment

One way to handle planning in a dynamic environment is to interleave planning and execution [14]. 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.

### *2.2.1 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 [16], and planners with explicit sensing operations such as SGP [3], Cassandra [18], and UWL [10]. C-BURIDAN [8], 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.

### 2.2.2 *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[4]. 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 use by the scheduler: cost, quality, and duration.

Once the Planner chooses the best tasks according to the utility function, it creates options 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. 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.

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 replanning). 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 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"[11].
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.)

In the next section we examine further two of these reasons to plan for choice for the scheduler. The DECAF agent planner implements to some degree all but the last reason for planning choices.

## 2.3 Creating Choice for the Scheduler

Consider a simple example where assembling Report3 requires finding the earnings per share (EPS) and price earnings ratio (PER) of a given company's stock. Some tasks associated with these goals are represented in Figure 5 as they are in the planner.

Action and task specifications contain syntactic predicate descriptions of the parameters and provisions they accept. The "effects" section of REPORT3TASK1 can be read "This task, when executed by the agent, will return a message that includes a value associated with the keyword "?text" that makes the predicate (REPORT3 ?symbol ?text) true." Similarly, the "provisions" section means that the agent calling must provide a value for the variable "?symbol" that is appropriate for the predicate. 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 [28]). 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]For the planner, the parameters are a vector of arbitrary size.

| | Planning without options | Planning with options |
|---|---|---|
| plan task under utility= $1/duration$ | 8.2 sec | 9.7 sec |
| execute plan 5 times | 4.9 sec | 4.9 sec |
| utility, first part | $U_{\text{EPSACTION1}} = .076$ | $U_{\text{EPSACTION1}} = .068$ |
| Alter utility fcn to $Q/(C+D)$ | | |
| time to re-plan | 8.0 sec | no re-plan |
| execute plan 5 times | 7.8 sec | 11.8 sec |
| utility, second part | $U_{\text{EPSACTION3}} = 6.7$ | $U_{\text{EPSACTION2}} = 5.6$ |
| utility, second part counting time for re-plan | $U_{\text{EPSACTION3}+replan} = 4.0$ | $U_{\text{EPSACTION2}} = 5.6$ |

**Figure 4: Planning Twice vs. Planning Once: In this experiment, an agent generated and ran plans. In the middle column, the branching option of the planner was turned off, so that the planner made a plan to suit the given utility function. When the function changed, the planner was called to re-plan for the new function. In the right column, the dissimilar-branching option was on, so that the planner could give the scheduler a choice to make at runtime (see Figure 2). Even though the alternate was not optimal, it was competitive, and resulted in increased utility when plan time was considered. See Figure 5 for the action descriptions.**

*task-name:* REPORT3TASK1
*effects:* (KNOW ?text (REPORT3 ?symbol ?text))
*provisions:* (KNOW ?symbol (REPORT3 ?symbol ?text))
*subObjectives:* (AND
(KNOW ?value (EPS ?company ?value))
(KNOW ?value2 (PER ?company ?value2)) )
*quality:* 90 *cost:* 40 *duration:* 30

*action-name:* EPSACTION1
*effects:* (KNOW ?value (EPS ?company ?value))
*provisions:*(KNOW ?company (EPS ?company ?value))
*preconditions:* (SYMBOL ?company ?symbol)
*quality:* 120 *cost:* 70 *duration:*25

*action-name:* EPSACTION2
*effects:* (KNOW ?amount (EPS ?symbol ?amount))
*provisions:*(KNOW ?company (EPS ?company ?value))
*quality:* 90 *cost:* 10 *duration:*60

*action-name:* EPSACTION3
*effects:* (KNOW ?amount (EPS ?symbol ?amount))
*provisions:*(KNOW ?company (EPS ?company ?value))
*quality:* 80 *cost:* 20 *duration:*40

**Figure 5: Example task descriptions and behavior profiles (see Figure 4). The behavior profile numbers are actually represented with features, MAX and MIN, but are single valued here for simplicity.**

The precondition in EPSACTION1 is a constraint that requires that the agent currently "know" the market symbol for the company in question. If this were a planning predicate, the Planner would construct a plan to satisfy the preconditions.

Given an objective (AND (KNOW ?text (Report3 IBM ?text))), the planner finds all templates in the Plan Library whose effects can unify with the objective. One plan generated with the dissimilar branching option is presented in Figure 2. Here, as in Figure 4, the utility function was $U = 1/duration$. The task with the shortest duration, EPSACTION1, is thus selected first.

The planner now chooses an alternative task to present to the scheduler using one of two strategies pre-selected by the agent: the alternative task can either be the one closest in utility to the best task 3, or (as done here) the task with a behavior profile most unlike that of the best task 2. The latter option may be attractive in a very dynamic environment, e.g. if failure of the first task chosen might suggest trying a very different approach. To find the most "different" task, the task with the greatest sum-squared-distance from the centroid of the tasks already chosen, in this case EPSACTION2.

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.

Figure 4 shows some timed results of planning without branching options versus planning for dissimilar task branching. While our experiment is somewhat artificial[5], it illustrates a result that we believe will become more pronounced and more useful as our plans become increasingly complex and our system resources become more precious.

## 3. RELATED WORK

Of the planners that plan for mutually exclusive contingencies (including CNLP [16], SGP [3], Cassandra [18], UWL [10], and C-BURIDAN [8]), 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 [28]. The DECAF planner currently lacks critics, in part because the information gathering tasks it currently 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 [15]. 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 [25] 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.

## 4. 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 also designs plans to make use of an "intelligent" scheduler, increasing the flexibility of every agent's plans and (we expect) decreasing the overall time spent in planning.

The simple problems the DECAF Planner has addressed so far have suggested that our approach has merit. We are currently expanding our library of actions and decompositions for information retrieval, as well as investigating economic simulation as a domain for further work.

## 5. REFERENCES


[1] R. Bonasso, D. Kortenkamp, and T. Whitney. Using a robot control architecture to aoutomate space shuttle operation. In *Proceding of the Ninth Conference on Innovative Appications of AI*, 1997.

[2] 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.


[5] Only the plan-time and run-time numbers are "hard"; and the utility functions were chosen to make a point.


[3] C. A. 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.

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

[5] K. S. Decker and V. R. Lesser. Quantitative modeling of complex computational task environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 217–224, Washington, July 1993.

[6] K. S. Decker and V. 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.

[7] K. S. Decker and K. Sycara. Intelligent adaptive information agents. *Intelligent Information Systems*, 9:239–260, 1997.

[8] 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.

[9] K. Erol, D. Nau, and J. Hendler. Semantics for hierarchical task-network planning. Technical report CS-TR-3239, UMIACS-TR-94-31, Computer Science Dept., University of Maryland, 1994.

[10] 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, Oct. 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[11] A. Garvey and V. 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.

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

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

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

[15] 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.

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

[17] C. J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, December 1996.

[18] L. Pryor and G. Collins. Planning for contingencies: a decision-based approach. *Journal of Artificial*

*Intelligence Research*, 4:287–339, 1996.

[19] A. Rao and M. 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.

[20] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, Mass., 1994.

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

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

[23] R. Simmons. Structured control for autonomous robots. *IEEE Trans. on Robotics and Automation*, 10(1), Feb. 1994.

[24] K. Sycara, K. S. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, Dec. 1996.

[25] M. Veloso, M. Pollack, and M. 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.

[26] 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.

[27] M. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proc. fo the First Intl. Conf. on Artificial Intelligence Planning Systems*, pages 236–242, June 1992.

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

[29] M. Williamson, K. S. Decker, and K. Sycara. Executing decision-theoretic plans in multi-agent environments. In *AAAI Fall Symposium on Plan Execution*, Nov. 1996. AAAI Report FS-96-01.

[30] M. Williamson, K. S. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

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

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