Efficient Meta-Level Control in Bounded-Rational Agents *

Anita Raja and Victor Lesser
Computer Science Department
University of Massachusetts

UMass Computer Science Technical Report 2002-051

December 3, 2002

Abstract

Complex agents operating in open environments must make real-time control decisions on scheduling and planning of domain actions. These decisions are made in the context of limited resources and uncertainty about outcomes of actions. The question of how to sequence domain and control actions without consuming too many resources in the process is the meta-level control problem for a resource-bounded rational agent. Our approach is to design and build a meta-level control agent architecture with bounded computational overhead. It supports decisions on when to accept, delay or reject a new task, how much effort to put into scheduling when reasoning about a new task and whether to reschedule when actual execution performance deviates from expected performance. We show that efficient meta-level control leads to significant improvement in performance and provide empirical results to support our claim.

* Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525 and by the National Science Foundation under Grant number IIS-9812755 and number IRI-9634938. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, NSF, USAF, or the U.S. Government.

Keywords: action selection and planning in agents; motivation, goal selection and theories of rational agency; agent architectures

1 Introduction

Agents in complex environments must reason about their local problem solving actions, interact with other agents, plan a course of action and carry it out. All these have to be done in real time in the face of limited resources and uncertainty about action outcomes. Furthermore, new tasks can be generated by the environment at any time, which in turn may necessitate rescheduling or replanning. This requires an agent's deliberation to be interleaved with execution. The planning and scheduling of tasks are non-trivial activities, requiring either exponential work, or in practice, a sophisticated scheme that controls the complexity. In this paper, we describe a meta-level control architecture which provide effective allocation of computation resulting in improved performance of individual agents in a cooperative multi-agent system

We classify agent actions into three categories - **domain**, **control**, and **meta-level control** actions. Domain actions are executable primitive actions that achieve the various high-level tasks. Control actions are scheduling actions that choose the high level tasks, set constraints on how to achieve them and sequence the detailed domain level actions that achieve the selected tasks. Other potential control actions are coordination actions that facilitate cooperation with other agents in order to achieve the high-level tasks; organizational adaptation and communication activities that generally occur in multi-agent systems. For the purposes of this paper we restrict control actions to just scheduling actions within a single agent and study the effect of optimizing the sequential decision making process. Meta-level control actions optimize the agent's performance by choosing and sequencing domain and control actions.

Agents perform control actions to improve their performance. Many efficient architectures and algorithms that support these actions have been developed and studied [1, 4, 5, 3, 9]. Agents receive sensations from the environment and respond by performing actions that affect the environment using their effectors. The agent chooses its domain level actions and this might involve invoking the scheduling module. Classic agent architectures either overlook the cost of control actions or they assume a fixed and negligible cost and do not explicitly reason about the time and other resources consumed by control actions, which may in fact degrade an agent's performance. An agent is not performing rationally if it fails to account for the overhead of computing a solution. This leads to actions that are without operational significance [6].

Consider an administrative agent capable of performing multiple tasks such as answering the telephone, paying bills and looking for information on laptops with the best value. It usually takes the agent a significant amount of time to find the laptop which best fits the user's preferences. Suppose the agent does not perform any meta-level reasoning about the importance or urgency of the tasks. It will then spend the same amount of time deciding whether to pick up a ringing phone as it does on deciding which laptop manufacturer sites to visit. If the agent is equipped with meta-level reasoning capabilities, it will recognize the need to make quicker decisions about the phone call than about the laptops since there is a tight constraint on the ringing phone, namely that the caller could hang up. Meta-level control will also allow the agent to dynamically change its decisions based on its current state. For instance, if the agent's deadline for determining the laptop information is imminent, the agent could decide not to answer any phone calls until the search is completed and the suggestion is made to the user. The agent is thus able to make better decisions about answering calls as well as completing its other tasks by dynamically adjusting its decision based on its current state and the tasks at hand.

Our agent architecture will support this dynamic adjustment process by introducing resource-bounded meta-level reasoning in agent control. Meta-level control actions allocate appropriate amount of processor and other resources to either domain or control actions at appropriate times. To do this optimally, an agent would have to know the effect of all combinations of actions ahead of time, which is intractable for any reasonably sized problem. The question of how to approximate this ideal of sequencing domain and control actions without consuming too many resources in the process, is the **meta-level control problem** for a resource bounded rational agent.

Our solution to this problem shows that a judicious choice of high-level features can be used by simple meta-level control rules to get significant increase in performance. The high-level state features provide qualitative characterizations of the system state. We also show that there is significant advantage to having a predictive model of task arrivals while making control decisions. To our knowledge this is the first demonstration of the effectiveness of meta-level control in complex agent architecture.

We construct a series of increasingly sophisticated approaches, all based on the abstract features used to represent system state, to handle the meta-level control problem. They differ by the amount of knowledge, including learned knowledge they use. In the most simple case, the heuristic policy is a set of hand-generated rules that are mostly environment independent. Next, we explore a set of more sophisticated set of hand generated rules that use knowledge about task characteristics including arrival times and deadlines. We compare these approaches to two baseline strategies: random and deterministic and show that the heuristic strategies perform significantly better than the baseline approaches in Section 5. The heuristic strategies provide a better baseline to evaluate learning strategies for meta-level control because they are more indicative of the positive effects of meta-level control.

We plan to use these abstract state features to represent the state of a MDP-based meta-level controller which uses reinforcement learning (RL). The abstract features bound the otherwise exponential state space of the MDP for this complex problem.

The paper is structured as follows: we enumerate the assumptions made in our approach in Section 2 and describe the agent architecture in which meta-level control will operate in Section 3. In Section 4, we present and evaluate a case-base of hand-generated heuristics for the different meta-level control decisions. Experimental results illustrating the strength of meta-level control in agent reasoning and the effectiveness of the heuristics are provided are given in Section 5. In Section 6, we describe the conclusions and the future directions of this work.

2 Assumptions

The following assumptions are made in this work: Each agent α has a finite set of tasks T_i which are generated by the environment and arrive in a finite interval of time. The overall goal of an agent is to maximize the utility generated over this finite time horizon. Each agent has a model of all the high-level tasks it is capable of performing. The agent is not explicitly aware of the arrival model of tasks but can potentially learn information that either implicitly or explicitly models the environment.

Each task T_i arriving at an agent has an *arrival time* AT_i and a *deadline* DL_i associated with it. An agent may concurrently pursue multiple high-level tasks and the agent derives utility by completing a task successfully within its deadline. It is not necessary for all high-level tasks to be completed in order for an agent to derive utility from its actions. A task T_i can be achieved by one of various alternative ways(plans) $P_i{}^j, P_i{}^{j+1}, P_i{}^{j+2}...P_i^k$. A plan $P_i{}^j$ is a sequence of executable primitive actions $P_i{}^j = \{m_1, m_2, ...m_n\}$ and has a *utility distribution* $UD_{P_i{}^j}$ and *duration distribution* $DD_{P_i{}^j}$ associated with it. The tasks do not accrue utility uniformly over their execution duration, instead they gain utility only when execution of the entire plan completes within the task deadline.

The agent's control decisions involve choosing which of these high-level tasks to pursue and how to go about achieving them. There can be local dependencies within the primitive actions belonging to a task. These dependencies can be hard or soft precedence relationships. Scheduling actions do not have to be done immediately after there are requests for them and in some cases may not be done at all. There are alternative ways of completing scheduling activities which trade off the likelihood of these activities resulting in optimal decisions versus the amount of resources used. System execution is single threaded allowing for one primitive action at the most to be in execution at any time.

3 Agent Architecture

In this section, we describe an open agent architecture which provides efficient meta-level control for bounded rational agents. Figure 1 describes the control flow in this architecture.

Environment: The environment consists of a task generator which generates tasks for individual agents based on an arrival model.

Meta-Level Control Layer (MLC): The MLC is invoked when certain exogenous or internal events occur. The controller computes the corresponding system state and determines the best action prescribed by the policy for that particular task environment. The policy is a simple hand-generated heuristic policy in the case of the naive heuristic strategy (NHS) and a more complex heuristic policy based on task arrival information in the case of the sophisticated heuristic strategy (SHS).

This architecture accounts for computational and execution cost at all three levels of the decision hierarchy: domain, control and meta-level control activities. The cost of domain activities is modeled directly in the task structures which describe the tasks. They are reasoned about by control activities like scheduling. Performance profiles of the various control activities are used to compute their costs and are reasoned about by the meta-level

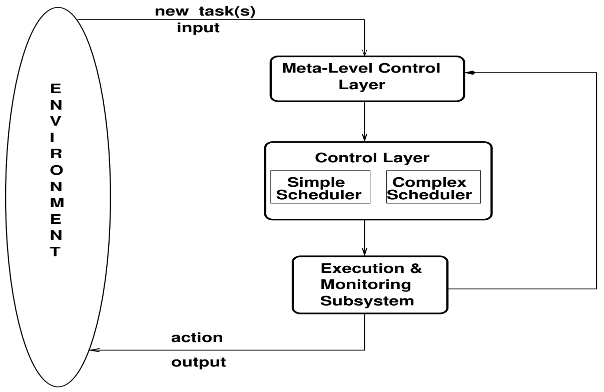

Figure 1: Control-flow in a bounded rational agent

controller. Meta-level control activities in this architecture are modeled as activities with small yet non-negligible costs which are incurred by the computation of state features which facilitate the decision-making process. These costs are accounted for by the agent, whenever events trigger meta-level activity. The state features and their functionality are described in greater detail in the next section.

The following are three events that are handled by the MLC and the corresponding set of possible action choices. *Arrival of a new task*: When a new task arrives at the agent, the meta-level control component has to decide whether to reason about it later; drop the task completely; or to do scheduling-related reasoning about an incoming task at arrival time and if so, what type of scheduling - complex or simple. The decision tree describing the various action choices named A1-A9 is shown in Figure 2. Each of the meta-level decisions has an associated decision tree. Scheduling actions have costs with respect to scheduling time and decommit costs of previously established commitments if the previous schedule is significantly revised or completely dropped. These costs are diminished or avoided completely if scheduling a new task is postponed to a later convenient time or completely avoided if the task is dropped. The meta-level controller can decide that it does not have enough information to make a good decision and will consequently choose to spend more time in collecting features which will help with the decision making process. The meta-level controller can hence choose to spend more resources to make a better informed decision.

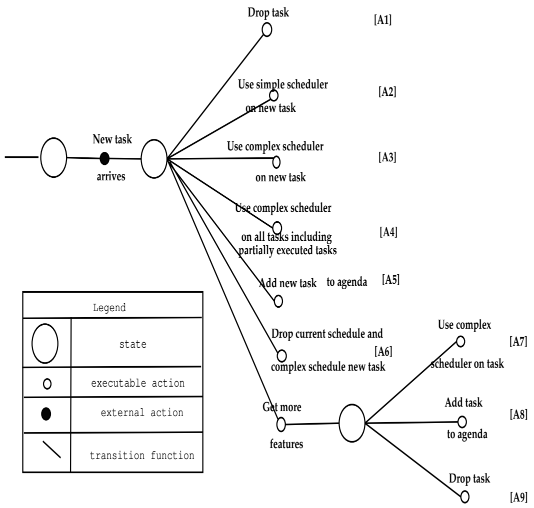

Figure 2: Decision tree when a new task arrives

Invocation of the detailed scheduler: The parameters to the scheduler are scheduling effort, time to schedule for and slack amount. They are determined based on the current state of the system including characteristics of the existing schedule and the set of new tasks that are being scheduled. The *scheduling effort* parameter determines the amount of computational effort that should be invested by the scheduler. The parameter can be set to either *HIGH*, where a high number of alternative schedules are produced and examined or *LOW*, where pruning occurs at a very early stage and hence few alternative schedules are compared, reducing the computational effort while compromising the accuracy of the schedule. The *time to schedule for* parameter determines the earliest starting time for the schedule to begin execution. This parameter is offset by the sum of the time needed to complete any primitive executions whose execution has been interrupted by the meta-level control action and the time spent on scheduling the new task(s). The *slack* parameter determines the amount of flexibility available in the schedule so that unexpected events can be handled by the agent without it detrimentally affecting its expected performance characteristics. The amount of slack to be inserted depends on two factors, the amount of uncertainty in the schedule as well the amount of expected meta-level control activity that will occur during the duration of the schedule. The scheduler determines the amount uncertainty in the schedules it builds and automatically inserts slack to handle highly uncertain primitive actions. The meta-level control component uses information about the arrival of future tasks to suggest slack amounts to the scheduler. This information is readily available by the sophisticated heuristic strategy. The naive heuristic approach uses a simple method of predicting arrival characteristics of future tasks based on past task arrival characteristics and is described in the next section.

Domain action completes execution: When a primitive action is completed, the MLC checks to see if the real-time performance of the current schedule is as expected. If the actual performance deviates from expected performance by more than the available slack time, then a reschedule may be initiated. A decision to reschedule helps in two ways: it would preclude the agent from reaching a bad state in which too many resources are spent on a schedule with bad performance characteristics; and it would allow for meta-level activities to be processed without the detrimental effects such processing would have on domain activities if slack is minimal.

Control Layer: The control layer consists of two schedulers, simple and complex schedule, which differ in their performance profiles.

Simple Scheduler: The simple scheduler is invoked by the MLC and receives the task structure and goal criteria as input. It uses the pre-computed abstract information of the task to select the appropriate schedule which fits the criteria. This will support reactive control for highly time constrained situations. When an agent has to schedule a task but doesn't have the resources or time to call the complex domain-level scheduler, the generic abstraction information of the task structure can be used to provide a reasonable but often non-optimal schedule.

The agent gathers knowledge about all tasks that it is capable of performing by subjecting each task through an abstraction process. Abstraction is an offline process where potential schedules and their associated performance characteristics for achieving the high level tasks are discovered for varying objective criteria. This is achieved by systematically searching over the space of objective criteria. The abstraction hides the details of these schedules and provides only the high level information necessary to make meta-level choices.

Complex Scheduler: The domain level scheduler depicted in the architecture is an extended version of the Design-to-Criteria (DTC) scheduler[8]. Design-to-Criteria (DTC) scheduling is the soft real-time process of finding an execution path through a hierarchical task network such that the resultant schedule meets certain design criteria, such as real-time deadlines, cost limits, and utility preferences. Casting the language into an action-selecting-sequencing problem, the process is to select a subset of primitive actions from a set of candidate actions, and sequence them, so that the end result is an end-to-end schedule of an agent's activities that meets situation specific design criteria. If the meta-level action is to invoke the complex scheduler, the scheduler component receives the task structure, objective criteria and a set of scheduler parameters as input and outputs the best satisficing schedule as a sequence of primitive actions.

Execution and Monitoring Layer:

The control layer can invoke the execution component either to execute a single control action prescribed by the meta-level controller or a series of domain actions determined by the control component. The execution results are sent back to the MLC where they are evaluated and if the execution performance deviates from expected performance, then a reschedule is initiated.

This architecture and control flow provides the agent the capability to adapt to changing conditions in an unpredictable environment. This is explained in greater detail in the next section, Moreover, the architecture is open in that the modules belonging to the various layers can be replaced by modules with better performance characteristics and the advantages of the architecture will still hold true.

4 Meta-level control

The meta-level controller uses the current state of the agent to make appropriate decisions. The real state of the agent has to account for every task which has to be reasoned about by the agent as well all the execution characteristics of each of these tasks. This makes the system state, even for simple scenarios, continuous and complex leading to an enormous state space. We handle the complexity of the real state by defining a higher level abstract state which captures the important qualitative state information relevant to the meta-level control decision making process. The following are some of the features of the abstract state used by the meta-level controller.

F1: Utility goodness of new task: It describes the utility of a newly arrived task based on whether the new task is very valuable, moderately valuable or not valuable in relation to other tasks being performed by the agent.

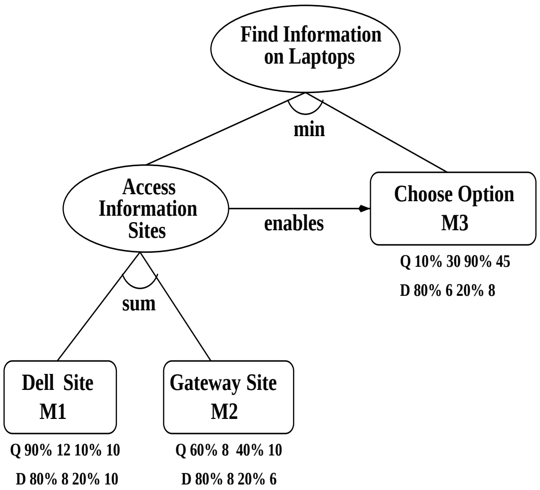

Figure 3: Task T1

F2: Deadline tightness of a particular task: It describes the tightness of the deadline of a particular task in relation to expected deadlines of other tasks. It determines whether the new task's deadline is very close, moderately close or far in the future.

F3: Utility goodness of current schedule: It describes the utility of the current schedule normalized by the schedule length and is based on information provided by the scheduler. This feature determines whether the current schedule is very valuable, moderately valuable or not valuable with respect to other tasks and schedules.

F4: Deadline tightness of current schedule: It describes the deadline tightness of the current schedule in relation to expected deadlines of tasks in that environment. It determines whether the schedule's deadline is very close, moderately close or far in the future.

F5: Arrival of a valuable new task: It provides the probability of a high utility, tight deadline task arriving in the near future by using information on the task characteristics like task type, frequency of arrival and tightness of deadline.

Each of the state features takes on qualitative values such as high, medium and low. The quantitative values such as utility of 80 versus utility of 60 are classified into these qualitative buckets (high versus medium utility) in a principled way as shown later in this section. As will be seen in the experimental results in Section 5, these qualitative measures provide information that can be exploited to make effective meta-level control decisions.

We use a simple example to describe how these features are computed, thereby determining the systems state. We then provide examples of heuristics which use the system state to choose the appropriate meta-level decision.

Suppose an agent A can perform the obtaining information on laptops(Task T1). Instances of this task and other tasks have associated arrival times and deadlines. Tasks are represented as task structures. Figure 3 is the task structure for *Obtain Information on Laptops* task. The top-level task is to *Obtain Information on Laptops.* The information gathering task is decomposed into a subtask which is to *Access Information Sites* and method *Choose Option* which compares information gathered from the various sites. Methods are primitive actions which can be scheduled and executed and are characterized by their expected utility and duration distributions. For instance, the utility distribution of method *Choose Option* described as 10% 30 90% 45, indicates that it achieves a utility value of 30 with probability 0.1 and utility of 45 with probability 0.9. Utility is a deliberately abstract domain-dependent concept that describes the contribution of a particular action to overall problem solving.

Obtain Information on Laptops can achieve utility only by completing subtask *Access Information Sites* successfully and then executing the method *Choose Option* successfully. The minimum of the utilities is propagated to the *Obtain Information on Laptops* task. This is indicated by the min() utility accumulation function(qaf), which defines how performing the subtasks relate to performing the parent task. The *enables* arc between *Access Information Site* and *Choose Option* is a non-local-effect (nle) or task interaction; it models the fact that information of laptops needs to be obtained from various manufacturer sites in order to perform comparisons on the various products. Subtask *Access Information Sites* can be achieve by successfully executing one or any combination of the methods and the maximum of the utilities of the methods which were executed is propagated back the subtask.

$T1^A$ ={M1,M2,M3} , $T1^B$ ={M1,M3} and $T1^C$ = {M2,M3} are three alternate plans to achieve task $T1$. The duration distribution of $T1^A$ is (13% 20 58% 22 26% 24 3% 26), which means that plan $T1^A$ takes 30 units of time 13% of the time, 22 time units 58% of the time and so on. Also $T1^A$ has a utility distribution of (6% 18 58% 20 36% 22). The utility distribution ($UD_{P_i^j}$) and duration distribution ($DD_{P_i^j}$) for each plan P_i^j is given below.

$$UD_{T1^A} = (13\%\ 20\ 58\%\ 22\ 26\%\ 24\ 3\%\ 26)$$

$$DD_{T1^A} = (6\%\ 18\ 58\%\ 20\ 36\%\ 22)$$

$$UD_{T1^B} = (10\%\ 10\ 90\%\ 12)$$
$$DD_{T1^B} = (64\%\ 14\ 32\%\ 16\ 4\%\ 18)$$
$$UD_{T1^C} = (60\%\ 8\ 40\%\ 10)$$
$$DD_{T1^C} = (16\%\ 12\ 68\%\ 14\ 16\%\ 16)$$

We now use an example to show how these low-level system parameters can be abstracted to determine the high-level system state, Suppose T1 arrives at time 45 and has a deadline of 62.

The *earliest start time* EST_i for a task T_i is the arrival time AT_i of the task delayed by the sum of R_{m_j}, the time required for completing the execution of the action m_j which is interrupted by a meta-level control event and C_i, the time required for scheduling the new task.

$$EST_i = AT_i + R_{m_j} + C_i$$

Suppose in this example there is no other task in execution when T1 arrives at time 45 and the average time for scheduling task T1 is 4 units. So, $EST_{T1} \ = \ 45 + 4 \ = \ 49$

The *maximum available duration* MD_i for a task T_i is the difference between the deadline of the task and its earliest start time.

$$MD_i = DL_i - EST_i$$

So, $MD_{T1} = 17$

Given a task T_i and its maximum available duration MD_i, the *probability that a plan P_i^j meets its deadline* $PDL_{P_i^j}$ is the sum of the probabilities of all values in the duration distribution of plan P_i^j which are less than the task's maximum available duration. For the above constraint where the maximum available duration for task $T1$ is 17,

$$PDL_{P_i^j} = \sum_{j=1}^{n} p_j : ((100 * p_j\%\ x_j)\ \epsilon\ DD_{P_i^j}) \wedge (x_j < MD_i)$$

$$PDL_{T1^A} = \ 0.0; \ \ PDL_{T1^B} = \ 0.64 \ + \ 0.32 = 0.96$$
$$PDL_{T1^C} = \ 0.16 \ + \ 0.68 \ + \ 0.16 = 1.0$$

The *expected duration* $ED_{P_i^j}$ *of a plan* P_i^j, is the expected duration of all values in the duration distribution of plan P_i^j which are less than the maximum available duration for the task.

$$ED_{P_i^j} = \frac{\sum_{j=1}^{n} p_j * x_j}{PDL_{P_i^j}} : ((100 * p_j\%\ x_j)\ \epsilon\ DD_{P_i^j}) \wedge (x_j < MD_i)$$

$$ED_{T1^A} = 0.0; \quad ED_{T1^B} = \frac{(64\% * 14 \ + \ 32\% * 16)}{0.98} = 14.37$$

$$ED_{T1^C} = \frac{(16\% * 12 \ + \ 68\% * 14 \ + \ 16\% * 16)}{1.0} = 14$$

The *expected utility* $EU_{P_i{}^j}$ *of a plan* $P_i{}^j$, is the product of the probability that the alternative meets its deadline and the expected utility of all values in the utility distribution of alternative $P_i{}^j$.

$$EU_{P_i{}^j} = \sum_{j=1}^{n} PDL_{P_i{}^j} * p_j * x_j : ((100 * p_j\% \ x_j) \ \epsilon \ UD_{P_i{}^j})$$

When the maximum available duration for task $T1$ is 17, $EU_{T1^A} = 0.0$

$$EU_{T1^B} = 0.98 * 0.1 * 10 \ + \ 0.98 * 0.9 * 12 = 11.564$$

$$EU_{T1^B} = 1.0 * 0.6 * 8 \ + \ 1.0 * 0.4 * 10 = 8.8$$

Given the maximum available duration for a task, the *preferred alternative* ALT_i for a task T_i is the alternative whose expected utility to expected duration ratio is the highest. ALT_i is the alternative which has the potential obtain the maximum utility in minimum duration within the given deadline.

$$ALT_i = P_i{}^j : \max_{j=1}^{n} \frac{EU_{P_i{}^j}}{ED_{P_i{}^j}}$$

Plan $T1^A$'s expected utility to expected duration ratio is 0, plan $T1^B$'s expected utility to expected duration ratio is $\frac{11.564}{14.37} = 0.804$ and plan $T1^C$'s expected utility to expected duration ratio is $\frac{8.8}{14} = 0.629$. So the alternative with the maximum expected utility to expected duration ratio is $T1^B$

$$ALT_{T1} = T1^B$$

The *utility goodness* UD_i(feature F1) of a task T_i is the measure which determines how good the expected utility to expected duration of the task's preferred alternative is in relation to the expected utility to expected duration ratio of the preferred alternatives of all the other tasks which arrive at the system. The tasks with high utility are the those whose expected utility to expected duration ratio are in the 66th percentile(top 1/3rd). Tasks whose ratios are in the middle third are tasks with medium utility and tasks whose ratios are in the bottom third are of low utility.

The utility goodness of the current schedule(feature F3) changes as execution of the schedule proceeds. The utility goodness increases as more effort is put into executing the plans which are part of the agent's schedule. This is because the tasks accrue utility only when their entire plan is completed and not in the midst of the plan execution.

The *deadline tightness*(feature F2) TD_i of a task T_i measures the flexibility of the maximum available duration. It determines the amount by which the maximum available duration of the can be reduced by unexpected meta-level actions and similar delays and the system without having a detrimental effect on the performance characteristics of the preferred alternative for the task.

In the interests of space, we do not describe the detailed computation underlying the determination of the tightness of deadline feature of the task, but it is similar to the determination of utility goodness.

The *deadline tightness of the current schedule*(feature F4) measures the flexibility in the execution duration of a schedule. The lower the flexibility, the tighter the deadline.

Suppose a meta-level action on average has an expected duration of C_{ML}. The *expected amount of time required for handling unexpected meta-level actions* CML_i, during the execution of task T_i, is computed as follows:

$$CML_i = C_{ML} * \frac{N_{CT}}{CT} * MD_i$$

This parameter is computed only by the meta-level controller using the naive heuristic strategy and is used to determine the amount of slack to insert into the schedule. Suppose the average time spent on a meta-level action is 2 units, 4 tasks have arrived at the agent and the current time is 45. MD_i is 17 as determined previously. $CML_{T1} = 2 * \frac{4}{45} * 17 = 3.02$ The amount of time expected to be spent on future meta-level actions is 3 units.

The *high priority task set for an agent* α $HPTS_\alpha$ is the set of tasks whose utility goodness is HIGH and deadline tightness is TIGHT.

$$HPTS_\alpha = \{T_k\} : (UG_k = HIGH) \wedge (TD_k = TIGHT)$$

The *arrival rate of high priority tasks* (feature F5) for an agent α, ART_α, is the ratio of the number of high priority tasks that arrive at the system to the total number of tasks n that have arrived at the system.

As mentioned in the previous section, there are three events which invoke the meta-level controller: arrival of a new task; invocation of the scheduler;

and a domain action completes execution. For the purposes of this discussion, we focus our attention on the decision making process involved with the arrival of a new task.

Naive Heuristic Strategy (NHS)

The NHS uses state-dependent hand-generated heuristics to determine the best course of meta-level control action. The current state information will allow the meta-level controller to dynamically adjust its decisions. The heuristics, however, are myopic and do not reason explicitly about the arrival of tasks in the near future.

The following are some of the heuristics used for decision-making by the NHS. In the interests of space, we have enumerate only a few interesting heuristics used to make a decision on a new task which has just arrived at the agent which allow the NHS to make efficient action choices.

Heuristic Rules for NHS:

- If new task has low utility goodness, tight deadline; current schedule is of high priority(high utility, tight deadline), then best action is *drop new task.*

- If new task has high priority; current schedule has low utility goodness, then best action is *drop its current schedule and schedule the new task immediately* independent of the schedule's deadline..

- If new task has low or medium utility goodness, tight deadline is to be scheduled, the agent will use the abstraction-based*simple scheduler.*

- If new task and current schedule are of high priority(high utility, tight deadline); then best action is *delay new task.* This is justified because the effort invested in the executing schedule makes it more valuable than a new task since the schedule is closer to accruing utility. This is a consequence of the task characteristic which accrues utility only when its plan is completely executed rather than accruing utility uniformly over its execution duration.

Sophisticated Heuristic Strategy (SHS)

The Sophisticated Heuristic Strategy(SHS) is a set of hand-generated rules which use knowledge about task arrival models to predict the environment characteristics. An environment is typically characterized by the expected utilities of the tasks, their deadline tightness and frequency of arrival. In this paper, we provide the information on the three parameters to the SHS. However, this information will be learned by the SHS by gathering statistics over multiple runs in future implementations. The meta-level controller can make non-myopic decisions by including information about its environment in its reasoning process. The following are some sample heuristics which show that the SHS can be more discriminatory about its decisions than NHS since it reasons about tasks that could arrive in the future.

Heuristic Rules for SHS:

- If new task has low utility goodness, tight deadline; high probability of high priority tasks arriving in the near future, then best action is *drop new task.*
- If new task has very low utility goodness, loose deadline; low probability of a high priority tasks arriving in the near future, then best action is *schedule new task using simple scheduling.*
- If new task has high priority; current schedule has low utility, tight deadline; low probability of high priority tasks arriving in the near future, then best action is*drop its current schedule and schedule the new task immediately.*
- If new task and current schedule are of high priority; high probability of high priority tasks arriving in the near future, then best action is *drop new task and continue with the current schedule*
- If a low utility new task is to be scheduled; low probability of a high priority task arriving in the near future, then best action is to use abstraction-based *simple scheduler* independent of the new task's deadline tightness.
- If new task has medium or low utility, medium or loose deadline; current schedule has high utility; low probability of high priority task arriving in the near future, then reasoning about the new task should be *delayed* till later.

5 Experimental Results

All the components of the agent architecture described in Section 3 have been implemented. This includes the meta-level controller, the various

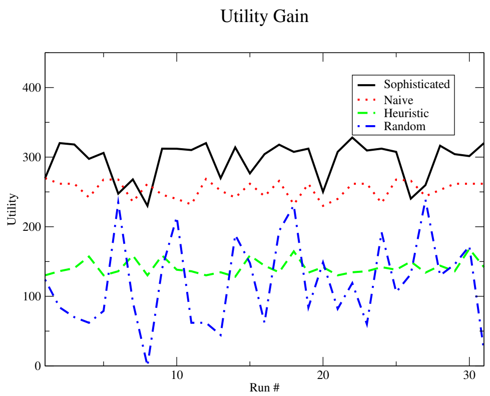

Figure 4: Experimental results comparing Utility Accrued by the four different strategies to meta-level control: Sophisticated Heuristic Strategy (SHS); Naive Heuristic Strategy (NHS); Deterministic Strategy; and Random Strategy over 30 runs each lasting 500 time units. The results are for an instance of an environment where tasks have medium arrival frequency and medium deadline tightness

schedulers with different performance profiles, and execution and monitoring component. The agent was built using the JAF agent framework [2]. In the experiments, we compare the performance of four different strategies to meta-level control:

- Sophisticated Heuristic Strategy (SHS): The SHS uses additional knowledge about the task arrival models to make non-myopic choices of meta-level control actions.

- Naive Heuristic Strategy (NHS): As described earlier, NHS uses myopic heuristics that do not explicitly reason about the likelihood of arrival of tasks in the near future.

- Deterministic Strategy: The deterministic strategy uses a fixed choice of meta-level action. When a new task arrives, this strategy always chooses to perform complex scheduling on the new task along with the tasks in the current schedule and tasks in the agenda. The schedule is always invoked with a fixed effort level of high and fixed slack amount of 10% of the total schedule duration. This strategy also does not

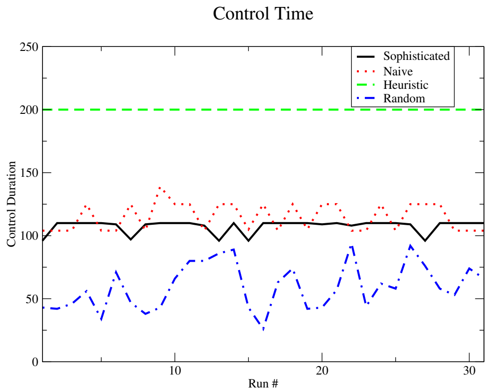

Figure 5: Experimental results comparing Control Durations by the four different strategies for an instance of the environment where tasks have medium arrival frequency and medium deadline tightness

reschedule upon when execution of a primitive completes independent of its performance characteristics.

- Random Strategy: The random strategy randomly chooses its actions for each of the three meta-level control decisions.

The task environment generator randomly creates task structures while varying three critical factors, namely, the complexity of tasks $c \epsilon \{simple, complex, combination\}$, frequency of arrival $f \epsilon \{high, medium, low\}$ and tightness of deadline $dl \epsilon \{tight, medium, loose\}$. Complexity of tasks refers to the expected utilities of tasks and the number of alternative plans available to complete the task. Typically, complex tasks have higher expected utility, higher expected durations and a greater number of alternatives than simple tasks. The frequency of arrival of tasks refers to the number of tasks that arrive within a finite time horizon. The contention for resources among the tasks increases as the task frequency increases. The tightness of deadline refers to the parameter defined in the previous section and it is task specific. The resource contention is also proportional to the deadline tightness.

The experimental results in this section show the performance of the various strategies for a particular environment. Each strategy was evaluated over 30 runs of the system and each run lasted 500 time units; The environment contains a combination of simple and complex tasks. The frequency

	SHS	NHS	Deterministic	Random
AUG	295.85 (± 27.29)	253.45 (± 13.09)	141.21 (± 11.10)	120.12(±64.83)
ACT	107.51 (± 5.07)	114.67 (± 11.35)	200.00 (± 0.0)	59.54 (± 18.40)
AT	12	12	12	12
ATC	6.7	6.0	4.0	2.64
PTC	55.83%	50%	33.33%	22%

Table 1: Each column represents each of the four algorithms; row 1 shows the average utility gain (AUG) and standard deviation per run; row 2 is the average control duration (ACT) and standard deviation per run; row 3 is the average number of tasks that arrive (AT) at the system; row 4 is the average number of tasks completed (ATC); row 5 is the percent of tasks completed (PTC)

of task arrival is medium and ranges between 10 and 15 tasks in the 500 time unit interval. The deadline tightness is also medium with respect to the expected task durations. Simple tasks have an average duration of 30 time units and complex tasks have an average duration of 55 time units. The simulation experiments were conducted using the Mass simulator [7].

Figure 4 shows the utility accrued over 30 runs by each of the four strategies. The heuristic strategies (SHS and NHS) significantly ($p < 0.05$) outperform the deterministic strategy. The accepted hypothesis is that SHS and NHS on average successfully completed at least 16% more tasks than the deterministic strategy. The random strategy is outperformed by the other three strategies.

The average utility achieved and the duration spent on control actions and their respective standard deviations and the percent of tasks completed by each of the four algorithms over 30 runs are shown in Table 1. The heuristic strategies complete at least 10% more tasks than the deterministic strategy.

The utility gained by SHS for this environment was significantly better than NHS. The null hypothesis of equivalence was also rejected at the .05 level when comparing the utilities of SHS and NHS. The SHS has access to critical information about the arrival model of the tasks and uses it to make non-myopic decisions about incoming new tasks, resulting in better utility gain. The NHS, on the other hand, does not have access to the information on arrival model and hence frequently makes meta-level decisions that have to be retracted because of the unexpected arrival of new tasks.

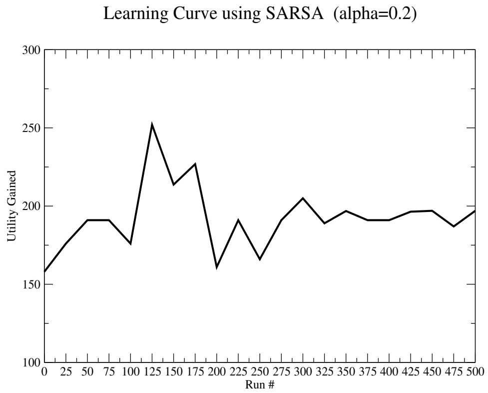

Figure 6: Learning curve for an instance of environment where tasks have medium frequency of arrival and medium deadline tightness

Figure 5 provides a comparison of the duration spent on control actions, We see that the heuristic strategies spend significantly less time on control actions that the deterministic method The huge control costs of the deterministic method limits the amount of time spent on domain actions, resulting in low utility gains.

Figure 6 shows the learning curve for the above described environment. We used SARSA, an online, on-policy reinforcement learning algorithm. This was a viable algorithm since the number of abstract states for the meta-level control MDP where less than 30. Each states had a total of 18 actions, only a subset of which were allowed at any particular state. We ran the learning algorithm for 500 episodes with epsilon of 0.2 for the first 300 episodes and then degrading to 0 after that. The learning rate parameter was set to 0.12. Each episode had an average of 15 decision points where the Q-values were updated and each episode took 4 minutes on a 1.8GHz Pentium(R) machine. The algorithm for the above parameters converged to a local maximum and we will continue to tweak the parameters till we can get the agent to learn the optimal policy for the complex single agent environment.

Figures 9 through 12, shown in the appendix, compare the utility gained by the four strategies for a number of different environments. It can be observed that the heuristic strategies always perform significantly better than the deterministic strategy. It can also be seen that in tightly constrained

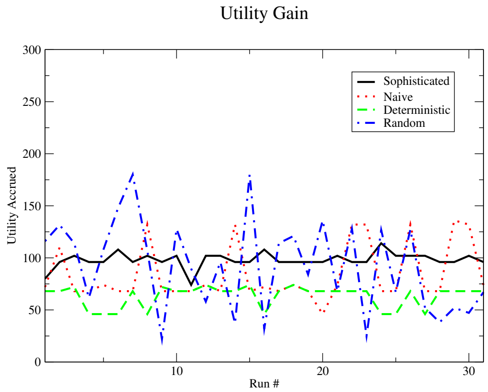

Figure 7: Experimental results comparing utilities gained by the four strategies for tasks with high frequency of arrivals and high deadline tightness

environments with high frequency of task arrivals and tight deadlines(see Figures 7), the difference in performance of the four strategies is not as pronounced as it is in environments with looser constraints (see Figure 8). This is because in tightly constrained environments, they number of tasks that can be completed successfully is limited and so no one strategy completely outperforms any other. Also, on average SHS was found to perform as well as if not significantly better than NHS among all the environments.

6 Conclusions and Future Work

In this paper we present a novel meta-level control agent architecture for bounded-rational agents. The meta-level control has limited and bounded computational overhead and will support reasoning about planning and scheduling costs as first-class objects. We have shown experimentally that meta-level control is beneficial. The heuristics described in this paper enable the meta-level controller to make satisficing decisions which adapt to different environments. The significance of the solution approach described in this paper comes from the following observation: A few abstract features which accurately capture the state information and task arrival model enable the meta-level control component to make useful decisions which significantly improve agent performance.

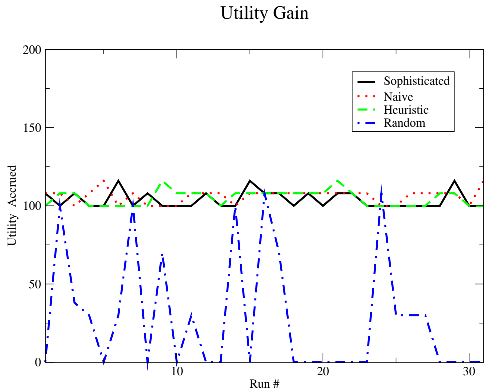

Figure 8: Experimental results comparing utilities gained by the four strategies for tasks with low frequency of arrivals and low deadline tightness

We plan to extend this work by introducing more complex features which will make the reasoning process more robust. Some such features include relation of slack fragments in local schedule to new task. This will enable an agent to fit a new task in its current schedule if it is possible and avoid a reschedule. Another feature would be to estimate the decommitment cost for a particular task. This will enable us to consider environments in which agents can decommit from tasks which they have previously agreed to complete. In this work, we assume that the arrival model is directly provided to the sophisticated heuristic approach. An extension to make the approach more feasible would be for the agent to learn the arrival model over multiple runs of the system and then use the appropriate heuristics.

As mentioned earlier, we plan to use the insight gathered from the heuristic approach to construct the state features, reward functions and algorithms to apply a reinforcement learning approach to this problem. We expect this analysis to provide valuable experience about applying RL techniques to complex real-world problems.

Our overall goal is to introduce efficient meta-level control in cooperative multi-agent systems. We believe that optimizing meta-level control in cooperative systems would lead to efficient meta-level control of the entire multi-agent system. We will begin by considering coordination, which facilitates cooperation with other agents in order to achieve high-level tasks, as a control action. Coordination and scheduling are tightly coupled control actions - coordination usually requires multiple calls to a scheduler. We plan to study the interactions between the different control actions and how this would affect the various meta-level control strategies.

7 Acknowledgments

We thank Bryan Horling for his assistance in extending the Mass simulator and Beenish Chaudry for her work on the parametrized environment generator.

References

[1] Craig Boutlier. Sequential Optimality and Coordination in Multiagent Systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[2] Bryan Horling and Victor Lesser. A reusable component architecture for agent construction. Master's thesis, Department of Computer Science, University of Massachusetts, Amherst, 1998. Available as UMASS CS TR-98-30.

[3] Kazuhiro Kuwabara. Meta-level control of coordination protocols. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS96)*, pages 104–111, 1996.

[4] David J. Musliner. Plan Execution in Mission-Critical Domains. In *Working Notes of the AAAI Fall Symposium on Plan Execution - Problems and Issues*, 1996.

[5] Anita Raja, Victor Lesser, and Thomas Wagner. Toward Robust Agent Control in Open Environments. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 84–91, Barcelona, Catalonia, Spain, July,. ACM Press.

[6] H. Simon. From substantive to procedural rationality, 1976.

[7] Regis Vincent, Bryan Horling, and Victor Lesser. Multi-agent system simulation framework. In *16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation.* EPFL, August 2000.

[8] Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.

[9] Shlomo Zilberstein and Abdel-Illah Mouaddib. Reactive control of dynamic progressive processing. In *IJCAI*, pages 1268–1273, 1999.

APPENDIX

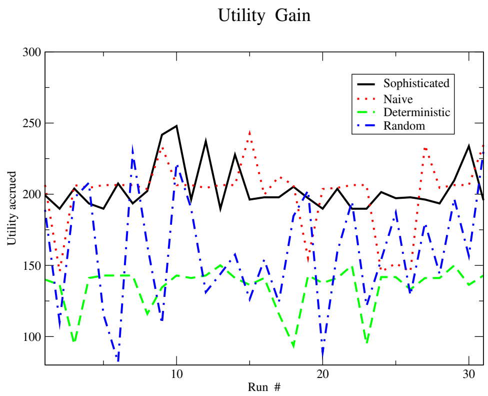

Figure 9: Experimental results comparing utilities gained by the four strategies for tasks with high frequency of arrivals and medium deadline tightness

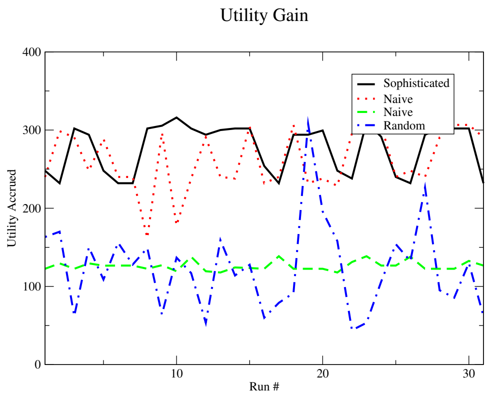

Figure 10: Experimental results comparing utilities gained by the four strategies for tasks with high frequency of arrivals and low deadline tightness

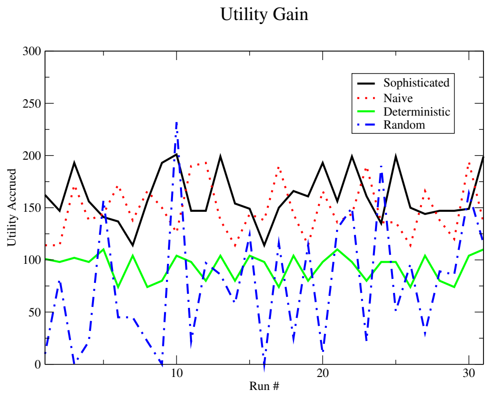

Figure 11: Experimental results comparing utilities gained by the four strategies for tasks with medium frequency of arrivals and high deadline tightness

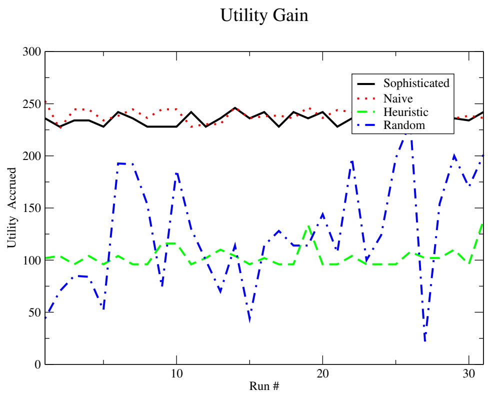

Figure 12: Experimental results comparing utilities gained by the four strategies for tasks with medium frequency of arrivals and low deadline tightness