Automated Meta-Level Control Reasoning in Complex Agents

Anita Raja and **Victor Lesser**
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003-4610, USA
araja,lesser@cs.umass.edu
Ph: 413-545-3444 Fax: 413-545-1249

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 based on hand-generated heuristics and reinforcement learning to support our claim.

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 provides 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.

Meta-level control actions optimize the agent's performance by choosing and sequencing control actions. The meta-level control problem is a sequential decision making process because it involves sequences of control decisions whose consequences emerge over time periods of variable and uncertain duration. The meta-level control decision strategy should thus take into account both the expected short-term and long-term consequences of its decisions.

Agents perform control actions to improve their performance. Many efficient architectures and algorithms that support these actions have been developed and studied [Boutlier, 1999; Musliner, 1996; Raja *et al.*, 2000; Kuwabara, 1996; Zilberstein and Mouaddib, 1999]. 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 control actions such as 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 [Simon, 1976].

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 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. Finally, we describe a how these abstract state features are used 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.

We compare the heuristic 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 the learning strategy for meta-level control because they are more indicative of the positive effects of meta-level control.

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 7, 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).

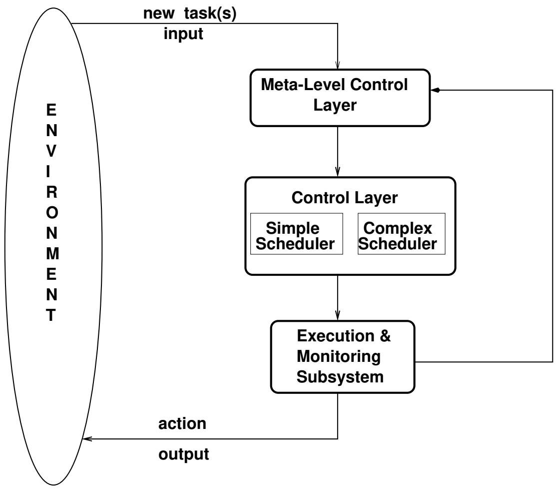

Figure 1: Control-flow in a bounded rational agent

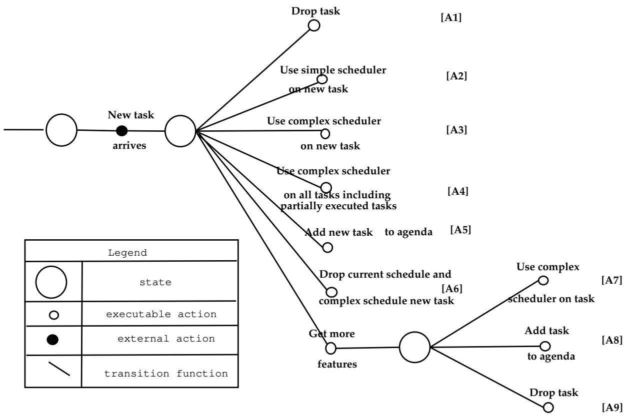

Figure 2: Decision tree when a new task arrives

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 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.

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 of 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 to 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 schedulers, 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 in the form of linear sequence of primitive actions and their associated performance characteristics such as expected quality distribution, expected duration distribution, and expected duration uncertainty 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[Wagner *et al.*, 1998]. 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 MLC in making its decisions does not directly use the information contained in the agent's current state. This would include detailed information regarding the tasks that are not yet scheduled, status of tasks that are partially executed, and the schedule of primitive actions that are to be executed. Instead the MLC uses in it decision making, a set of high-level qualitative features that are computed from the full state information and pre-computed information about the behavior of the tasks that the system can handle. The advantage of this approach is that it simplifies the decision making process and provides the possibility for learning these rules (which we are currently exploring). The following are some of the features of the high-level state used by the meta-level controller. They take on qualitative values such as high, medium and low.

F1: Utility goodness of new task 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.

F2: Deadline tightness of a new task describes the tightness of the deadline of a new 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 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 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 describes 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.

We will now describe some of the low-level parameters that determine the high-level features of the system state. Suppose an agent *A* can perform task T which is *Obtain Information on Laptops*. Figure 3 describes how this task is decomposed into a subtask which is to *Access Information Sites* and method *Choose Option* that compares information gathered from the various sites. Methods are primitive actions that 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.

T^A ={M1,M2,M3}, T^B ={M1,M3} and T^C = {M2,M3} are three alternate plans to achieve task T. The duration distribution of T^A is (13% 20 58% 22 26% 24 3% 26), which means that plan T^A takes 30 time units 13% of the time, 22 time units 58% of the time and so on. Also T^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.

$$
\begin{aligned}
UD_{T^A} &= (13\%\ 20\ 58\%\ 22\ 26\%\ 24\ 3\%\ 26)\\
DD_{T^A} &= (6\%\ 18\ 58\%\ 20\ 36\%\ 22)\\
UD_{T^B} &= (10\%\ 10\ 90\%\ 12)\\
DD_{T^B} &= (64\%\ 14\ 32\%\ 16\ 4\%\ 18)\\
UD_{T^C} &= (60\%\ 8\ 40\%\ 10)\\
DD_{T^C} &= (16\%\ 12\ 68\%\ 14\ 16\%\ 16)
\end{aligned}
$$

We now use an example to show how these low-level system parameters can be used to determine the high-level features of the system state. Suppose T arrives at time 45 and has a deadline of 66.

Definition 1: 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 T arrives at time 45 and the average time for scheduling task T is 4 units. So, $EST_T = 45 + 4 = 49$

Definition 2: 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, $MD_T = 17$

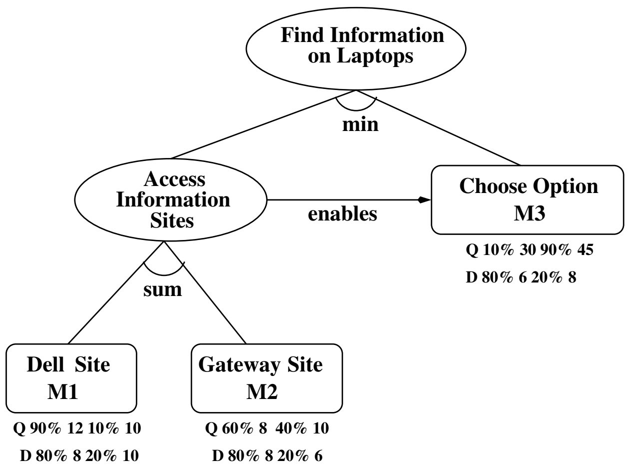

Figure 3: Task T

Definition 3: 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 ${P_i}^j$ that are less than the task's maximum available duration. For the above constraint where the maximum available duration for task T is 17,

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

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

Definition 4: 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 ${P_i}^j$ that are less than the maximum available duration for the task.

$$
\begin{aligned}
ED_{P_i{}^j} &= \frac{\sum_{j=1}^{n} p_j * x_j}{PDL_{P_i{}^j}} : ((100 * p_j\%\ x_j)\ \epsilon\ DD_{P_i{}^j}) \wedge (x_j < MD_i)\\
ED_{T^A} &= 0.0\\
ED_{T^B} &= \frac{(64\% * 14\ +\ 32\% * 16)}{0.98} = 14.37\\
ED_{T^C} &= \frac{(16\% * 12\ +\ 68\% * 14\ +\ 16\% * 16)}{1.0} = 14
\end{aligned}
$$

Definition 5: 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 T is 17,

$$
\begin{aligned}
EU_{T^A} &= 0.0\\
EU_{T^B} &= 0.98 * (0.1 * 10\ +\ 0.9 * 12) = 11.564\\
EU_{T^C} &= 1.0 * (0.6 * 8\ +\ 0.4 * 10) = 8.8
\end{aligned}
$$

Definition 6: 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 that 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 T^A's expected utility to expected duration ratio is 0, plan T^B's expected utility to expected duration ratio is $\frac{11.564}{14.37} = 0.804$ and plan T^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 T^B and $ALT_T = T^B$

Definition 7: The *utility goodness* UD_i(feature F1) of a task T_i is the measure that 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 that arrive at the system. The tasks with high utility are 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.

Definition 8: 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 actions the agent's schedule.

Definition 9: 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 task 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. The detailed computation underlying the determination of the tightness of deadline feature of the task is omitted in the interests of space, but it is similar to the determination of utility goodness.

Definition 10: 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}.

Definition 11: 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: $CML_i = C_{ML} * \frac{N_{CT}}{CT} * MD_i$ where N_{CT} is the total number of tasks that have arrived at the system from the start to current time CT. 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.

Definition 12: 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. $HPTS_\alpha = \{T_k\} : (UG_k = HIGH) \wedge (TD_k = TIGHT)$

Definition 13: 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.

We now describe the heuristic strategies which use hand-generated rules for meta-level control based on the above mentioned high-level features. The heuristic strategies are meant to provide a sanity check on the usefulness of these features.

4.1 Heuristic Strategies

The heuristic strategies are discussed in the context of the decision making process when a new task arrives at the agent. The agent has similar heuristic rules for the other two events: invocation of the domain scheduler and domain action completion.

The **Naive Heuristic Strategy (NHS)** uses state-dependent hand-generated heuristics based on high-level features to decide what control decision to make. These heuristics are myopic and do not reason explicitly about the arrival of tasks in the near future. The following are some of the heuristics used to make a decision when a new task arrives.

- 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.

The **Sophisticated Heuristic Strategy(SHS)** is a set of hand-generated rules that use knowledge about environment characteristics to make non-myopic decisions. An environment is typically characterized by qualitative information on the quality the tasks that will arrive during the episode , their deadline tightness and frequency of arrival.

The knowledge of the task arrival model enables the SHS to make decisions that in the long term minimize the resources spent on redundant control actions. For instance, in the first example heuristic rule described below, if the meta-level controller is aware that tasks with high expected utilities and tight deadlines will arrive frequently, it will decide to drop a new task with low utility and tight deadline because there is a high probability that the decision to schedule it would be reversed when a high priority new task arrives in the near-future. Similarly if the meta-level controller is aware that task arrival is infrequent, it would make a decision to schedule the low priority task because there is a high probability that the decision will not be reversed and the agent will gain some utility instead of just waiting for the infrequent high priority task.

We provide the environment characteristics directly to the SHS. However, this information can be learned. The following are some sample heuristics that show that the SHS can be more discriminatory about its decisions than NHS since it reasons about tasks that could arrive in the future.

- 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*.

4.2 Reinforcement Learning

In the Reinforcement Learning (RL) framework, the learning agent interacts with an environment over a series of time steps t=0,1,2,3... At each time t, the agent observes the environment states, s_t, and chooses an action, a_t, which causes the environment to transition to state s_{t+1} and to emit a reward, r_{t+1}. In a Markovian system, the next state and reward depend only on the preceding state and action, but they may depend on these in a stochastic manner. The objective of the agent is to learn to maximize the expected value of reward received over time. It does this by learning a (possibly stochastic) mapping from states to actions called a *policy*. More precisely, the objective is to choose each action a_t so as to maximize the expected return, $E(\sum_{i=1}^{\infty} \gamma^i r_{t+i+1})$, where $\gamma \in [0, 1)$ is a discount parameter. A common solution strategy is to approximate the optimal action-value function, or Q-function, which maps each state and action to the maximum expected return starting from the given state and action and thereafter always taking the best actions.

We adopt the learning approach developed in [Singh *et al.*, 2000] for using RL in the design of a spoken dialogue system. Their problem is similar to ours in that it is also a sequential decision making problem and there is a bottle neck associated with collecting training data. Each of our simulation runs takes approximately four minutes since we are accounting for real-time control costs.

As described earlier, the MLC in making its decisions does not directly use the information contained in the agent's current state. The state of the markov-decision process is an abstraction of the actual state of the systems and uses the features described previously. We then specified the appropriate actions to take in each state. The actions are a list of control actions. Action nodes A1-A9 Figure 2 describe some of the action choices. The reward function is the sum of the utilities accrued by each completed task. The meta-level control policy is a mapping from each state to an action.

We then implemented an initial meta-level control policy which randomly chooses an action at each state and collects a set of episodes from a sample of the environment. Each episode is a sequence of alternating states, actions and rewards. As described in [Singh *et al.*, 2000], we estimated transition probabilities of the form $P(s'|s,a)$, which denotes the probability of a transition to state s', given that the system was in state s and took action a from many such sequences. The transition probability estimate is the ratio of the number of times in all the episodes, that the system was in s and took a and arrived at s' to the number of times in all the episodes, that the system was in s and took a irrespective of the next state. The Markov decision process (MDP) model representing system behavior for a particular environment is obtained from state set, action set, transition probabilities and reward function. The efficiency of the model depends on the extent of exploration performed in the training data with respect to the chosen states and actions. In the final step we determine the optimal policy in the estimated MDP using the Q-value version of the standard value iteration algorithm [Sutton and Barto, 1998]. The expected cumulative reward (or Q-value) Q(s,a) of taking action a from state s is calculated in terms of the Q-values of successor states via the following recursive equation [Sutton and Barto, 1998]:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

The algorithm iteratively updates the estimate of $Q(s,a)$ based on the current Q-values of neighboring states and stops when the update yields a difference that is below a threshold. Once value iteration is completed, the optimal meta-level control policy (according to the estimated model) is obtained by selecting the action with the maximum Q-value at each state. To the extent that the estimated MDP is an accurate model of the particular environment, this optimized policy should maximize the reward obtained in future episodes. The summary of the proposed methodology is as follows

1. Choose an appropriate reward measure for episodes and an appropriate representation for episode states.
2. Build an initial state-based training system that creates an exploratory data set. Despite being exploratory, this system should provide the desired basic functionality.
3. Use these training episodes to build an empirical MDP model on the state space.
4. Compute the optimal meta-level control policy according to this MDP.
5. Reimplement the system using the learned meta-level control policy

5 Experimental Results

The meta-level controller, the various schedulers with different performance profiles, and the execution and monitoring component described in Section 3 have been implemented. In this section, we compare the performance of four different strategies to meta-level control: Naive Heuristic Strategy (NHS); Sophisticated Heuristic Strategy (SHS); Deterministic Strategy; and Random 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 scheduler is invoked with a fixed effort level of high and fixed slack amount of 10% of the total schedule duration. This strategy also does not reschedule when a primitive action completes execution. The random strategy randomly chooses its actions for each of the three meta-level control decisions.

Row#		SHS	NHS	Deter.	Rand.
1	AUG	205.49	192.10	121.90	89.97
2	σ	57.0	62.5	62.55	59.114
3	CT	20.37%	23.92%	39.27%	11.77%
4	RES	0%	14.53%	0%	50.56%

Table 1: Performance evaluation of four algorithms over a single environment with a combination of tasks, medium frequency of arrival and medium deadline tightness; Column 1 is row number; Column 2 describes the various comparison criteria; Columns 3-6 represent each of the four algorithms; Rows 1 and 2 show the average utility gain (AUG) and respective standard deviations (σ) per run; row 3 shows the percentage of the total 500 units spent on control actions(CT); row 4 is percent of tasks rescheduled (RES)

The task environment generator randomly creates task structures while varying three critical factors, namely, the complexity of tasks $c \in \{simple(S), complex(C), combination(A)\}$, frequency of arrival $f \in \{high(H), medium(M), low(L)\}$ and tightness of deadline $dl \in \{tight(T), medium(M), loose(L)\}$. 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 which 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 described in Table 1 show the performance of the various strategies in an environment which contains a combination of simple and complex tasks. The frequency of task arrival is medium and ranges between 9 and 13 tasks in the 500 time unit interval. The deadline tightness is also medium. Simple tasks have an average duration of 22 time units and complex tasks have an average duration of 32 time units. Each strategy was evaluated over 100 runs and each run has an associated task arrival model, lasts 500 time units and has an average of 15 meta-level control decision points per run.

Rows 1 and 2 of the table describe the average utility gained (AUG) by each of the strategies and the corresponding standard deviations. The heuristic strategies (SHS and

Environment#	SHS	NHS	Deter.	Rand.
AMT	117.34	117.25	82.17	67.33
AHT	123.35	122.29	60.27	76.72
ALM	135.05	124.74	115.93	48.21
CMM	163.77	157.27	103.33	50.86

Table 2: Utility Comparisons over a number of environments; Column 1 is the environment type; Columns 2-5 represents the average utility gained by each of the four algorithms for that environment;

NHS) significantly ($p < 0.05$) outperform the deterministic and random strategies with respect to utility gain. The accepted hypothesis is that SHS and NHS on average achieved at least 68.5% and 57.58% higher utility than the deterministic strategy respectively.

SHS has about a 10% improvement in utility gain than NHS. Upon detailed analysis of the data, we find that NHS assigns incorrect amounts of slack in the schedule which is required to handle unexpected meta-level activities. This leads to frequent reschedule calls and an increase in time spent on control actions. The SHS is able to allocate accurate amounts of slack because it has access to the task arrival model information and is able to avoid unnecessary control actions (particularly reschedules).

Row 3 shows the percent of the 500 time units for each run that was spent on control actions (CT) and row 4 shows the percent of tasks that were rescheduled (RES) per run in the midst of their execution. For the above mentioned reason, NHS has a significant number of reschedules resulting in time being spent on control actions instead of being spent the utility deriving domain actions. Row 3 shows that the duration spent on control actions by NHS is significantly ($p < 0.05$) higher than that of SHS. The deterministic strategy does not ever reschedule but invests a lot of time on control actions since the fixed strategy is time-intensive. The random strategy spends the least amount of time on control (11.77%) because it attempts relatively few tasks (there is a high probability of a task being dropped randomly upon arrival).

Tables 2 and 3 show the utility and control comparisons over a number of environments. Environment AMT means the complexity of tasks are combination (A), the arrival frequency is Medium (M); and the deadline tightness is Tight (T). The experiments in this section show that heuristic strategies provide a better baseline to evaluate other strategies for meta-level control, including learning strategies, as they are more indicative of the positive effects of meta-level control.

Our initial results described in Figure 4 shows the utility accrued and control time used by the reinforcement learning, SHS and NHS strategies for 2 environments AMM and AMT. The training data for the RL strategy consisted of 2000 episodes. The results are averaged over 300 simulation test episodes. The RL strategy with 100 training episodes was significantly better($p < 0.05$) than SHS with respect to utility and had significantly lower control duration. For the two environments studied, we see that performance improvement is not proportional to number of training episodes. The learning curve saturation properties are currently being studied.

Environment#	SHS	NHS	Deter.	Rand.
AMT	24.95%	20.32%	36.59%	8.07%
AHT	35.26%	28.98%	55.04%	16.63%
ALM	10.11%	10.32%	14.42%	4.61%
CMM	11.08%	10.99%	12.39%	4.29%

Table 3: Control Time Comparisons over a number of environments; Column 1 is the environment type; Columns 2-5 represents the % of total time spent on control actions by each of the four algorithms for that environment;

	RL-2000	RL-1000	SHS	NHS
AMM-UTIL	211.85	211.56	205.49	192.10
AMM-CT	18.14%	18.16%	20.37%	23.92%
AMT-UTIL	145.11	140.57	117.34	117.25
AMT-CT	17.31%	17.58%	24.95%	20.32%

Table 4: Utility and Control Time Comparisons over 2 environments; Column 1 is the environment type; Column 2 represents the performance characteristics of the RL policy after 2000 training episodes; Column 3 represents the performance characteristics of the RL policy after 1000 training episodes; Column 4 and 5 represent the performance characteristics of SHS and NHS respectively;

6 Related Work

There has been enormous amount of work on intelligent agent control [Musliner *et al.*, 1995; Garvey and Lesser, 1996; Wagner *et al.*, 1998; Boddy and Dean, 1994; Zilberstein and Russell, 1996]. These systems describe systems describe flexible and goal-directed mechanisms capable of recognizing and adapting to environmental and dynamics and resource bounds. The emphasis in these works is to build an adaptive control layer which reasons about domain-level costs. They, however, do not explicitly reason about the control costs. Our work reasons about control costs as first class objects and includes reasoning about costs at all levels of computation.

There has also been a lot of work on layered architectures. In the mid-1980's, the subsumption architecture [Brooks, 1986] was introduced. The Beliefs-Desires-Intentions (BDI) model [Bratman, 1987] is probably the most popular approach towards the design of intelligent agents, mainly due to trigger behaviors driven by conceptually modeled interactions and goals rather than procedural information. In addition, it seems to be a functional abstraction for the higher-level reasoning processes such as action selection. [Bratman *et al.*, 1991] describes Intelligent Resource-bounded Machine Architecture (IRMA), an architecture for resource-bounded (mainly in terms of computational power) deliberative agents, based on the BDI model. IRMA agents consist of four main modules: a means-end planner, an opportunity analyzer, a filtering process and a deliberation procedure. In addition, they contain a plan library and data structures to store beliefs, desires and intentions. The Procedural Reasoning System [Georgeff and Lansky, 1987] is a hybrid system, where beliefs are expressed in first-order logic and desires represent system behaviors instead of fixed goals. PRS agents are capable of deliberative and reactive control. They however do not have an explicit meta-level control component as described in our architecture. They focus on choosing deliberative or reactive control rather than balancing control and domain actions.

There has been previous work on meta-level control [Simon, 1976; Russell *et al.*, 1993; Stefik, 1981; Durfee and Lesser, 1988] but in reviewing the literature there is little that is directly related the meta-level control problem discussed in this paper. The difficult characteristics of the our problem are the complexity of the information that characterize system state; the variety of responses with differing costs and parameters available to the situation; the high degree of uncertainty caused by the non-deterministic arrival of tasks and outcomes of primitive domain actions; and finally the fact that the consequence of decisions are often not observable immediately and may have significant down-stream effects.

[Harada and Russell, 1999] describes initial work where the computational process is explicitly modeled. The paper describes initial ideas for using search as the model of computation in the Tetris domain. They propose the use of MDP's and reinforcement learning as their solution approach. This work was not pursued further(personal communication with second author). Our methodology was developed independently of their effort and was built for a more complex sequential decision making process involving a number of decisions.

The problem worked on that is closest to the complexity of our meta-level control decisions is the Guardian system[Hayes-Roth *et al.*, 1994]. However their system is knowledge intensive and the heuristic rules seem very domain-dependent in comparison to the domain-independence of our approach. The other work that seems directly applicable is [Russell and Wefald, 1992] and [Hansen and Zilberstein, 1996]. However the techniques used by both are limited to specific problem solving situations that were much more structured than those encountered in our domain.

[Lagoudakis and Littman, 2000] describes a reinforcement learning based algorithm for dynamically selecting the right algorithm for a given instance based on instance features while minimizing overall execution time. This problem has several interesting overlaps with our work although they only deal with a single problem instance at any point in time and do not deal with a sequential decision process which complicates the reasoning process.

7 Current 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 high-level 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. We have also shown that Reinforcement learning is potentially a viable approach for studying real-work, complex sequential decision making problems.

We plan to extend this work by introducing more complex features which will make the reasoning process more robust. The agent will be augmented with the capability to learn the task arrival model knowledge used by SHS. We are currently using insight gathered from the heuristic approaches and reinforcement learning approaches to study parameters which will allow for policy generalization over similar environments. We are are also experimentally studying the parameters for learning curve saturation for each environment. We expect this analysis to provide valuable experience about applying Reinforcement Learning techniques to complex real-world problems. And finally, we plan to reason about coordination, organizational adaptation and communication as control actions to achieve our overall goal of introducing efficient meta-level control in cooperative multi-agent systems.

References

[Boddy and Dean, 1994] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments, 1994.

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

[Bratman *et al.*, 1991] Michael E. Bratman, David Israel, and Martha Pollack. Plans and resource-bounded practical reasoning. In Robert Cummins and John L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1–22. The MIT Press, Cambridge, Massachusetts, 1991.

[Bratman, 1987] M.E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[Durfee and Lesser, 1988] E. Durfee and V. Lesser. Predictability vs. responsiveness: Coordinating problem solvers in dynamic domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 66–71, 1988.

[Garvey and Lesser, 1996] Alan Garvey and Victor Lesser. Issues in design-to-time real-time scheduling. In *AAAI Fall 1996 Symposium on Flexible Computation*, November 1996.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *In Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, WA*, pages (2) 677–682, 1987.

[Hansen and Zilberstein, 1996] Eric Hansen and Shlomo Zilberstein. Monitoring the Progress of Anytime Problem-Solving. In *Proceedings of the 13th National Conference* *on Artificial Intelligence*, pages 1229–1234, Portland, Oregon, 1996.

[Harada and Russell, 1999] Daishi Harada and Stuart Russell. Extended abstract: Learning search strategies. In *Proc. AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information, Stanford, CA, 1999.*, 1999.

[Hayes-Roth *et al.*, 1994] B. Hayes-Roth, S. Uckun, J.E. Larsson, D. Gaba, J. Barr, and J. Chien. Guardian: A prototype intelligent agent for intensive-care monitoring. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1503–1511, 1994.

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

[Lagoudakis and Littman, 2000] Michail G. Lagoudakis and Michael L. Littman. Reinforcement learning for algorithm selection. In *AAAI/IAAI*, page 1081, 2000.

[Musliner *et al.*, 1995] D. J. Musliner, J. A. Hendler, A. K. Agrawala abd E. H. Durfee, J. K. Strosnider, and C. J. Paul. The challenges of real-time ai. In *IEEE Computer*, pages 28(1):58–66, 1995.

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

[Raja *et al.*, 2000] 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, 2000. ACM Press.

[Russell and Wefald, 1992] S. Russell and E. Wefald. *Do the right thing: studies in limited rationality*. MIT press, 1992.

[Russell *et al.*, 1993] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 338–344, 1993.

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

[Singh *et al.*, 2000] Satinder P. Singh, Michael J. Kearns, Diane J. Litman, and Marilyn A. Walker. Empirical evaluation of a reinforcement learning spoken dialogue system. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 645–651, 2000.

[Stefik, 1981] M. Stefik. Planning and meta-planning. *Artificial Intelligence*, 16(2):141–170, 1981.

[Sutton and Barto, 1998] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[Wagner *et al.*, 1998] 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.

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

[Zilberstein and Russell, 1996] Shlomo Zilberstein and Stuart J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.