Leveraging Meta-level Control in Multi-Agent Systems

Anita Raja
Department of Software and Information Systems
The University of North Carolina at Charlotte
Charlotte, NC 28223
anraja@uncc.edu

Victor Lesser
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
lesser@cs.umass.edu

Abstract

Sophisticated agents operating in open environments must make complex real-time control decisions on scheduling and coordination of domain activities. These decisions are made in the context of limited resources and uncertainty about the outcomes of activities. Many efficient architectures and algorithms that support these computation-intensive activities have been developed and studied. However, none of these architectures explicitly reason about the consumption of time and other resources by these activities, which may degrade an agent's performance. The problem of sequencing execution and computational activities while reasoning about resource contention in the process, is the meta-level control problem for a resource-bounded rational agent.

In this paper we show that meta-level control with bounded computational overhead allows complex agents to solve problems more efficiently than current approaches in dynamic open multi-agent environments. We also show that meta-level control is computationally feasible through the use of an abstract representation of the agent state. This abstraction concisely captures critical information necessary for decision making while bounding the cost of meta-level control and is appropriate for use in automatically learning the meta-level control policies.

Keywords: Multi-Agent Systems, Bounded Rationality

1 Introduction

How does an agent efficiently trade-off the use of its limited resources between deliberations about which domain actions to execute and the execution of these domain actions? This is the meta-level control problem for agents operating in resource-bounded multi-agent environments.

Open environments are dynamic and uncertain. Complex agents operating in these environments must reason about their local problem solving actions, coordinate with other agents to complete tasks requiring joint effort, plan a course of action and carry it out. These deliberations may involve computation and delays waiting for arrival of appropriate information. They have to be done in the face of limited resources, uncertainty about action outcomes and in real-time. Furthermore, new tasks can be generated by existing or new agents at any time. These tasks have deadlines where completing the task after the deadline could lead to lower or no utility. This requires meta-level control which interleaves an agent's deliberation with execution of its domain activities.

Meta-level control involves deciding which deliberative actions to perform when, and whether to deliberate or to execute domain actions that are the result of previous deliberative actions. In this paper, deliberative actions are also referred to as control actions. If significant resources are expended on making this meta-level control decision, then meta-meta-level decisions have to be made on whether to spend these resources on meta-level control. However, if the meta-level reasoning process has a small computational overhead, there is no need for explicit meta-meta level reasoning. This avoids infinite regress of the meta-level control problem and is the approach used here. Meta-level control can be viewed as a sequential decision problem. The essence of sequential decision problems is that decisions that are made now can have both immediate and long-term effects; the best current action choice depends critically on the types of future situations the agent will face and the action choices which have to be made at those future decision points. The resource-bounds of the agent cause the current meta-level action choices to affect the resources available to future action choices. Effective meta-level control needs to use past performance information to make predictions about the future so as to make non-myopic decisions at each decision making point. This is in contrast to myopic decision making which tries to optimize only the next state of the system. The question of how to approximate this ideal selection and sequencing of domain and control actions without significant computational effort is the primary focus of this paper. Some of the characteristics of the meta-level problem that make it difficult are the complexity of the information that characterizes the state of the agent and other agents it interacts with; variety of responses with differing costs and parameters available to the situation; deadlines associated with these tasks; high degree of uncertainty caused by the non-deterministic arrival of tasks and outcomes of primitive domain actions; consequence of decisions are often not observable immediately and may have significant down-stream effects. To our knowledge, meta-level control for such a complex agent environment has not been studied.

In our work, each agent will have its own meta-level control component and the meta-level control policy will be computed offline. The cooperative nature of the problem environment in our work, is in that each agent has its individual goals to achieve and some of these goals require cooperation of other agents. The agents are trying to maximize the sum of the utilities attainable by the multi-agent system as a whole. The complex agents with individual meta-level control capabilities in this cooperative environment will give rise to meta-level control for the entire multi-agent system.

The three classes of deliberative actions discussed in this paper are: information gathering actions, planning/scheduling actions and coordination actions. The first type of deliberative action is information gathering which can be of two kinds. The first kind of information gathering involves gathering information about the environment which includes the state of other agents. This information is used by the meta-level controller to determine the relevant control actions. These actions do not use significant local processor time but they delay the meta-level deliberation process because of the end-to-end delay of getting information from other agents. The second kind of information gathering is the determination of complex state features of the agent which involve significant amount of computation. These features, for instance, can compute detailed timing, placement and priority information about the primitive actions which have to be executed to complete the agent's tasks. The agent must make explicit meta-level control decisions on whether to gather complex features and determine which complex features are appropriate. The second type of deliberative action involves planning and scheduling. Planning is the process in which the agent uses beliefs about actions and their consequences to search for solutions to one or more high-level tasks(goals) over the space of possible plans. It determines which domain actions should be taken to achieve the tasks. Scheduling is the process of deciding when and where each of these actions should be performed. In this paper, planning is folded into the scheduling. The agent's scheduling decisions involve choosing which of these high-level goals to pursue and how to go about achieving them. There can be local and non-local dependencies between tasks and methods. Local dependencies are inter-agent while non-local dependencies are intra-agent. These dependencies can be hard or soft precedence relationships. The meta-level control decision is to decide whether to invoke a scheduler, which scheduler to invoke, and how much resources to invest in the scheduling process. Finally, the third type of deliberative action, coordination, is the process by which a group of agents achieve their tasks in a shared environment. In this research, coordination is the inter-agent negotiation process that establishes commitments on finish times of tasks or methods done by one agent in the context of constraints of another agent's activities. The meta-level control decisions on coordination involve choosing the tasks that require coordination, deciding whether to coordinate with another agent and how much effort much be spent on coordination. Planning/Scheduling and coordination activities 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 planning/scheduling and coordination activities which trade-off the likelihood of these activities resulting in optimal decisions versus the amount of resources used. Agents make the simplifying assumption that results of coordination are binding and assume that other agents will not decommit from their commitments at later stages.

A problem with most multi-agent systems and agents [4, 28, 32, 21, 53] is that they do not explicitly reason about the cost of deliberative computation because they assume all deliberative computations are always done and always done in the same way. Thus, most systems, have no way to trade-off the resources used for deliberative actions and domain actions. An agent is not performing rationally if it fails to account for all the costs involved in achieving a desired goal. Failure to account for all costs could potentially lead to agents taking actions that are without operational significance [39]. Taking the entire cost of computation into account leads to what Simon calls procedural rationality, Good refers to as type II rationality [12] and what Russell and Wefald refer to as bounded rationality [33]. An agent exhibits such bounded rationality if it maximizes its expected utility given its computational and other resource limits.

We believe that as we build more advanced agents operating in less predictable real-time environments, reasoning about agent activities from this perspective will be crucial for effective agent operation. One of the characteristics of such advanced agents, that we will exploit in this paper, is that they have alternative ways of performing their deliberative computations that trade-off the use of fewer resources for a decrease in the optimality of decisions generated by the deliberative computation. These decisions involve choosing an order to solve goals, determining the allocation of resources that have to be dedicated to solving each of the goals, and establishing the role of other agents in facilitating the solution process. The optimality of making these decisions affects the total utility achieved by the group of cooperating agents. The high-level goals are generated either by sensing internal event triggers or by receiving requests for assistance from other agents. These goals must often be completed by a certain time in order to achieve any utility. It is not necessary for all high-level goals to be completed in order for an agent to derive utility from its activities, and partial satisfaction of a high-level goal is sometimes permissible while trading-off the amount of utility derived for decrease in resource usage.

The following assumptions are made in this paper: The agents are cooperative and will prefer alternatives which increase social utility/quality even if it is at the cost of decreasing local utility. An agent may concurrently pursue multiple high-level goals and completing a goal derives utility for the system or agent. The overall objective of the system or agent is to maximize the utility generated over some finite time horizon. Although a fixed horizon is used in the experiments, this information is not provided to the agents. This was deliberately done to equip the agents to operate in domains and environments with indefinite horizons (an unknown finite horizon).

The intent of this paper is to show that a meta-level reasoning component with bounded and small computation overhead can be constructed that significantly improves the overall performance of agents in a cooperative multi-agent system. Further, we show that appropriately abstracting the agent state is key top the development of the meta-level control component and that such abstraction can be the basis for automatically learning meta-level control policies.

The paper is structured as follows: We describe a formal model of the problem with an emphasis on the sequential decision making process that is involved. We then describe the difficulty in using this formal model as is which motivates our approximate solution method which capitalizes on the ability to model and use an abstract representation of the state. The following section describes the meta-level agent architecture which can support reasoning about costs at all levels of the decision making process; various meta-level decisions that need to be made; and the state information necessary to make these decisions. A description of high-level features that capture the state information concisely while bounding the size of the state space is also provided. A detailed example illustrating the functionality of the infrastructure is presented.

We then describe two strategies based on hand-generated heuristics: the naive heuristic strategy and the sophisticated heuristic strategy. They differ in the amount of environmental information available as part of the system state. These strategies use the high-level features that will be provided to the meta-level learning strategy. Snapshots of the meta-level reasoning process for specific exogenous events are also presented. The performance of the hand-generated strategies provide a sanity check on the effectiveness of the state features to allow for effective meta-level control.

Based on the positive results of the previous section, we show that a reinforcement learning strategy based on abstract state can be used to learn meta-level control policies within a reasonable number of learning episodes that produces policies that that are as effective as those that are hand-generated.

In the concluding sections of the paper we focus on the related work and how our approach differs significantly from other previous work. Specifically, we will discuss the seminal work of Russell and Wefald and how our work is different from their approach. We then conclude the paper with a review of the important ideas presented in the paper and experimental results and briefly discuss future work. We include a detailed trace of an agent operating with meta-level control as an appendix to the paper.

2 The Model

The following is a formal perspective of the meta-level control problem as described in this paper:

1. Let S be the set of states of the agent and $s_i \epsilon S$ is a particular state of the agent. Since this is a finite horizon problem, $i = 0, 1, 2, 3 \ldots, n$
2. A is the set of possible control actions and $a \epsilon A$ is the action taken by the agent in state s_i.

Control actions do not directly affect the utility achieved by the agent since they affect only the agent's internal state. These actions consume time and have only indirect effects on the external world.

Control actions are followed by the execution of utility achieving domain actions. These domain actions are directly the result of control actions in the current and preceding states. These domain actions are not explicitly represented in this model since they are encased by the control actions.

3. A policy π is a description of the behavior of the system. A stationary meta-level control policy $\pi : S \rightarrow A$ specifies, for each state, a control action to be taken. The policy is defined for a specific environment.

   An environment is defined by three distributions describing arriving task type, task arrival rate and task deadline tightness. The agents are situated in the context of other agents and tasks can be generated by the external environment which includes these other agents. Meta-level control is the decision process for choosing and sequencing control actions. In this work, there are five event triggers which invoke the meta-level control process. The occurrence of any of the triggers interrupts any other activity the agent in currently engaged in and control is shifted to the meta-level controller.

4. $\pi(s_i, a)$ is the probability of an agent taking an action a in state s_i under policy π.

5. s_j is the new state reached after executing control action a followed by the execution of corresponding domain actions that follow a.

6. $R(s_i, a, s_j)$ is the reward obtained in state s_j as a consequence taking control action a in state s_i and then executing the domain actions that follow a.

   The reward is the cumulative value of the tasks and domain actions which are completed between the state transitions. Since the values achieved by the tasks have associated uncertainties, the reward function is represented as a distribution.

7. $U_\pi(s_i)$ is the utility of state s_i under policy π.

8. $P(s_j|s_i, a)$ is the probability that agent is in state s_j as a result of taking action a in state s_i.

   The above model defines a finite Markov decision process [2].
   According to decision theory, an optimal action is one which maximizes the agent's expected utility, given by

$$E[U_\pi(s_i)] = E_\pi\{\sum_{j=1}^{n} \gamma^j \; R(s_i, a, s_j)\}$$

$\gamma \epsilon [0, 1)$ is a discount-rate parameter which determines the present value of future utility gains.
This can be computed as follows

$$E[U_\pi(s_i)] = \sum_a \pi(s_i, a) \sum_{j=1}^{n} P(s_j|s_i, a))[R(s_i, a, s_j) + \gamma \; E[U_\pi(s_j)]]$$

The meta-level control problem for an individual agent is to find a best meta-level control policy π^* which maximizes the expected return for all states. This optimal policy can be found using dynamic programming [2] and reinforcement learning [44] methods. These methods will implicitly determine the transition probability model and reward function defined previously. If the model were to be used as defined above, finding the optimal meta-level control policy would be computationally intractable because of the size of the state space. The quantitative values of the agent's state features contributes to the explosion of the state space. We feel that independent of the approach used to formulate meta-level control, the ability to appropriately abstract the agent state is key to the development of an effective meta-level control component.

We will now elucidate the meta-level control process by describing the situations in which meta-level control reasoning is required.

Taxonomy of meta-level decisions

There are five types of event triggers that require meta-level decision making in our framework.

1. Arrival of a new task from other agents or the external environment.
2. Presence of a task in the current task set that requires negotiation with a non-local agent.

3. Failure of a negotiation to reach a commitment.
4. Domain action completes execution requiring a check to see if there is a significant deviation of online schedule performance from expected performance.
5. Decision to schedule a new set of tasks or to reschedule existing tasks.[1]

These particular event triggers were chosen because they occur frequently in the domain described in this paper and the decisions made at these triggers affect the performance of the multi-agent system.

In order to provide a clear picture of these five decisions described above, consider the simple scenario consisting of two rovers *R1* and *R2*. Rovers are unmanned vehicles equipped with cameras and a variety of scientific sensors for the purpose of planetary surface exploration. The discussion here will focus on the various meta-level questions that will have to be addressed by *R1*. Figure 1 describes *Analyze Rock*, also called task *T0*, and *Explore Terrain*, also called task *T1*, which are the tasks performed by *R1*.

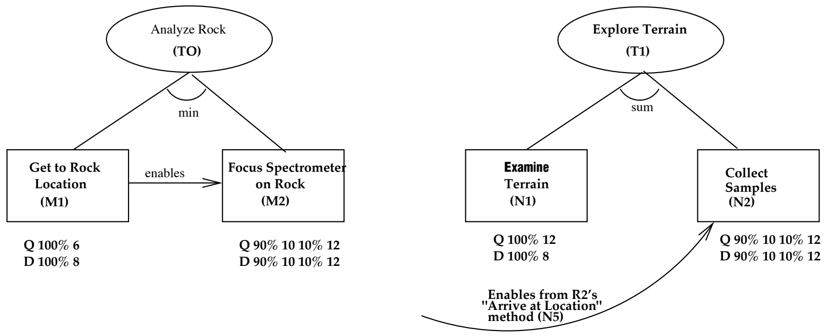


Figure 1: Tasks that can be performed by agent *R1*

In this example, each top-level task, as described in the TÆMS ask description language [7], is decomposed into two executable primitive actions. In order to achieve the task *Analyze Rock* , *R1* must execute both primitive actions *Get To Rock Location* and *Focus Spectrometer on Rock* in sequence. All primitive actions in TÆMS called *methods*, are statistically characterized in three dimensions: quality, cost and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Thus, different applications have different notions of what corresponds to model quality. The sequence is denoted by the enables arrow between the two actions and the min quality attribution factor (which denotes a conjunction operator) states that the minimum of the qualities of the two actions will be attributed to the *Analyze Rock* task. To achieve the task *Explore Terrain*, *R1* can execute one or both primitive actions *Examine Terrain* and *Collect Samples* within the task deadline and the quality accrued for the task will be cumulative (denoted by the *sum* function).

R2 is equipped with a storage compartment while R1 is not. The *Collect Samples* method requires R1 and R2 to coordinate: R1 has the ability to pick up the soil sample and put it in R2's storage compartment. This relationship between the two agents is denoted by the non-local *enables* from R2's *Arrive at Location Method (N5)* method (*R2's* task structure is not shown) to R1's *Collect Samples* method. Utility and duration distributions for each primitive action are provided.

The following are some of the meta-level questions that will be addressed by any individual agent. Each meta-level question is followed by a description of *R1's* perspective of each question and the associated costs and benefits. The state information and the trade-offs that affect the decision making process are enumerated.

1. Arrival of a new task from the environment.
   Meta-Level Question: Should *R1* schedule newly arriving task at arrival time or postpone scheduling to sometime in the future or drop that particular instance of the task.
   Benefit: If the new task has low expected utility, its deadline is very close and there is a high probability of a high utility task arriving in the future, then it should be discarded. This means *R1* chooses not to expend its limited resources on a low priority task and instead will wait for a future high priority task. If the incoming task has very high priority, in other words, the expected task utility is very high and it has a relatively close deadline, then *R1*

[1]This meta-level control decision is triggered as a consequence of one of the four decisions mentioned above.

should override its current schedule and schedule the new task immediately. If the current schedule has average utility that is significantly higher than the new task and the average deadline of the current schedule is significantly closer than that of the new task, then reasoning about the new task should be postponed till later.
Cost: There is the cost associated with the meta-level control decision. This is a small, fixed cost as described in the next section. Additionally, if the new task is scheduled immediately regardless of its expected utility or deadline, then its possible that the opportunity cost[2] of scheduling that task can be very high. Scheduling the new task has an associated cost in time units in addition to costs for dropping established commitments if the previous schedule is significantly revised or completely dropped. These costs can be diminished or avoided completely if the decision about the new task is postponed to later or completely avoided if the task is dropped.

2. Presence of a task in the current task set that requires negotiation with a non-local agent.
   (a) **Meta-Level Question:** Should method *Collect Samples*, which is enabled by *R2's* method *Arrive at Location*, be included in *R1's* schedule?
   Benefit: If method *Collect Samples* is included in *R1's* schedule, *R1* can increase its total utility.
   Cost: The small, fixed cost of the meta-level control decision. Additionally, if *R1* and *R2* have to negotiate over the finish time of *R2's* method *Arrive at Location*, there is a time cost.
   (b) **Meta-Level Question:** If *R1* decides to negotiate, it should also decide whether to negotiate by means of a single step or a multi-step protocol [23] that may require a number of negotiation cycles to find an acceptable solution or even a more expensive search for a near-optimal solution. For example, should the agent use a single shot protocol that is quick but has a chance of failure or a more complex protocol which takes more time, has a higher rate of success and often provides a better solution.
   Benefit: If *R1* receives high utility as a result of completing negotiation on finish time of *Arrive at Location*, then better the protocol, the higher the probability that the negotiation will succeed.
   Cost: The small, fixed cost of the meta-level control decision. Additionally, the protocols which have a higher guarantee of success require more resources, more cycles and more end-to-end time in case of multi-step negotiation and higher computation power and time in case of near-optimal solutions. (The end-to-end time is proportional to the delay in being able to start task executions).

3. Failure of a negotiation to reach a commitment.
   Meta-Level Question: If the negotiation between *R1* and *R2* using a particular negotiation protocol fails, should *R1* retry the negotiation with *R2* again[3]; if so, should *R1* use the same negotiation mechanism as before or an alternate mechanism; and how many such retries should take place?
   Benefit: Negotiation is preferred if *R1* will receive high utility as a result of R2 completing *Arrive at Location* method. There is a higher chance of negotiation succeeding if more time(cycles) is given. Since resources have been spent on figuring out a solution to the negotiation, it may be profitable to put in a little more effort to achieve a solution.
   Cost: The small, fixed cost of the meta-level control decision. Additionally, if *R1* and *R2* choose to negotiate over the finish time of R2's method *Arrive at Location*, this will take time and computational resources. If there is a very slight or no probability of finding an acceptable commitment, then resources which can be profitably spent on other solution paths are being wasted and the agent might find itself in a dead-end situation with no resources left for an alternate solution.

4. Decision to schedule a new set of tasks or to reschedule existing tasks.
   Meta-Level Question: When *R1's* scheduler is called, it has to decide how much effort to invest in scheduling. Also how flexible should the schedule produced by the detailed scheduler be? How much slack should be inserted in the schedule?
   Benefit: If the expected schedule performance characteristics are highly uncertain, then it is better for the scheduler to put in less effort. If there is slack in the schedule, then the system can deal with unanticipated events easily

[2]Resources that could have been invested in future high priority tasks are instead invested in lower priority tasks leading to lower overall utility gains.

[3]An interesting extension to this meta-level question which has not been explored in this paper is whether *R1* should renegotiate with *R2* or with another agent *R3* who is capable of performing the same task. The states of some, if not all the agents could have changed between the time of consideration of the previous negotiation and the current renegotiation and it might be better for *R1* to negotiate with *R3* rather than *R2*.

without having to bear the overhead of a reschedule.
Cost: The small, fixed cost for meta-level control. Additionally, if the schedule performance is not as expected, then rescheduling has to be called anyways and *R1* might put itself in a dead-end with no resources to find an alternate solution. This means a trade-off on the number of reschedule calls and effectiveness of these calls has to be made. Inserting slack in the schedule means that fewer primitive actions will be scheduled causing the available time to not be used to maximum capacity, but at the same time avoiding reschedule costs in case of unexpected events. Here, the resources allocated toward current goals is being traded-off to provide resources for unanticipated future events.

5. Significant deviation of online schedule performance from expected performance.
Meta-Level Question: When *R1's* schedule deviates from expected performance by threshold α, should a reschedule be invoked automatically?
Benefit: If *R1* observes that the schedule will fail to achieve its goal in a timely fashion, then it can reschedule and try an alternate path instead of going down a path which will definitely fail.
Cost: The small, fixed cost for meta-level control. Additionally, there is a computational cost associated with calling the scheduler and revising the commitments from the previous schedule that may lead to not finding a viable solution within a given deadline.

3 Meta-Level Control

Meta-level control is the process of optimizing an agent's performance by choosing and sequencing domain and control activities. First, a classic agent architecture augmented with a meta-level control component is presented. This includes a description of the interaction among the various components in the architecture and the agent's ability to reason about control costs as first class entities. A high-level representation of the state which captures the critical information while bounding the computation required to process the state is also described. Finally, a detailed example describing the functionality of an agent equipped with meta-level control reasoning capabilities is presented.

Agent Architecture

We will describe the role of meta-level control in the agent architecture by concentrating on the control flow among the various components(see Figure 2). In this architecture, the control components such as the schedulers, negotiation components and execution subsystem interact with the meta-level control (MLC) component. The MLC is invoked when certain exogenous or internal events occur (e.g. the request by another agent to perform a task for it). Both the meta-level and control components are involved in the agent decision making process. There are a number of data structures which help keep track of the agent's state. The NewTask List contains the tasks which have just arrived at the agent from the environment. The Agenda List is the set of tasks which have arrived at the agent but the reasoning about how to achieve the tasks has been delayed. They have not been scheduled yet. The Schedule List is the set of high-level tasks chosen to be scheduled and executed. The Execution List is the set of primitive actions which have been scheduled to achieve the high-level tasks and maybe in execution or yet to be executed. Examples of the decision making process corresponding to particular agent states are provided later in the section.

The meta-level is invoked when a new task arrives at the agent, even if the agent is in the midst of executing another task. The execution subsystem is invoked whenever the agent has to act upon the environment. These actions may or may not have immediate rewards. When an action completes execution, the execution subsystem sends the execution characteristics to the meta-level controller which is also the monitoring subsystem.

The control layer consists of schedulers and negotiation protocols. The two schedulers, simple and complex, differ in their performance profiles. Additionally, the complex scheduler is parametrized so the effort level and amount of slack inserted into a schedule can be varied.

Simple Scheduler: The simple scheduler is invoked by the meta-level controller and receives the task structure and goal criteria as input. It selects the most appropriate schedule for the current context from a set of pre-computed task schedules. This choice does not take into account other tasks that could be simultaneously scheduled with this task. When an agent has to schedule a task but doesn't have the resources or time to call the complex domain-level scheduler, the pre-computed information about the possible schedules 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 executing by performing off-line analysis on each task. This off-line process constructs potential schedules in the form of linear sequence of primitive actions. Each sequence has associated performance characteristics such as expected quality distribution, expected duration distribution, and expected duration uncertainty for achieving the high level tasks.

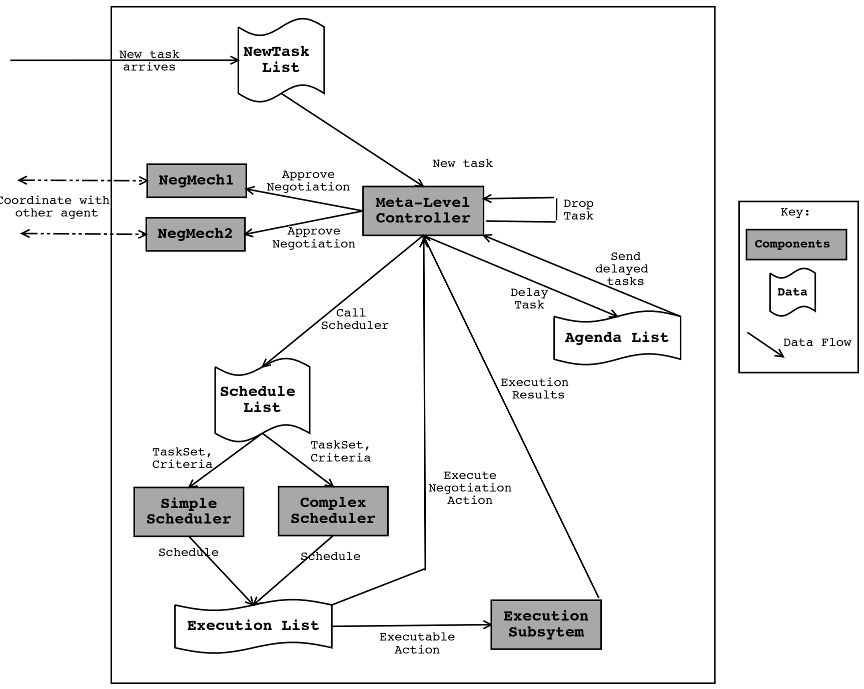


Figure 2: Control Flow in Meta-Level Control Agent Architecture

These performance characteristics are discovered by systematically searching over the space of objective criteria. The task abstraction hides the details of these schedules and provides only the high level information necessary to make meta-level choices.

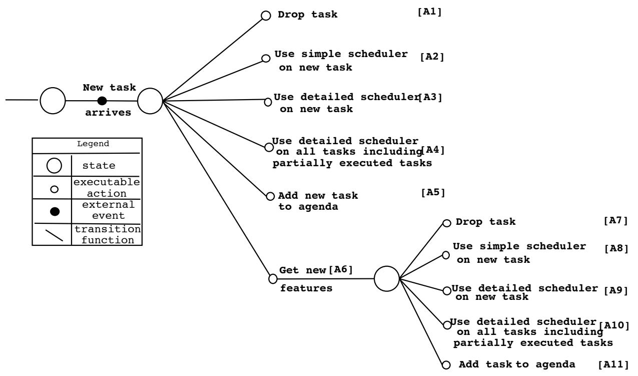
Complex Scheduler: The domain level scheduler depicted in the architecture is an extended version of the Design-to-Criteria (DTC) scheduler [50]. Design-to-Criteria (DTC) scheduling is the soft real-time process of finding an execution path through a hierarchical task network such that the resultant 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 a satisficing schedule as a sequence of primitive actions. The complex scheduler, in contrast to the simple scheduler, can reason about and schedule multiple tasks simultaneously. A detailed description of the scheduler parameters are provided later on in this section.

Negotiation Protocols: There are two types of negotiation protocols [23]: **NegMech1** and **NegMech2**. The choice of the exact negotiation protocol will depend on the relative gain of doing the associated task and the likelihood of the other agent doing the task. NegMech1 is a single-shot negotiation protocol that works in an all or nothing mode. A single proposal is sent out and single response is received. It is inexpensive but has a lower probability of success than the other negotiation protocol. NegMech2 is a multi-step negotiation protocol which tries to achieve a commitment by a sequence of proposals and counter-proposals until a consensus is reached or time runs out. It is more expensive than the single-shot protocol because of the computation and communication overhead. It, however, has a higher probability of success.

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. Domain activities 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 five events that are handled by the MLC and the corresponding set of possible action choices. Each of the external events and corresponding meta-level decisions has an associated decision tree. The external action triggers a state change. The response actions, execution of domain action or complex feature computation, are also modeled in the decision tree.

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 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-A8 is shown in Figure 3. 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 by adding it to the agenda of unscheduled tasks [A5] or completely avoided if the task is dropped [A1]. If a task is of high priority relative to other tasks in execution or on the agenda, the meta-level controller might decide to use the complex scheduler to schedule the task [A3]. If the new task is of high priority and the currently executing schedule is also of high priority, the meta-level controller could decide to reschedule all the tasks using the detailed scheduler [A4]. If there are tight constraints on scheduling the task, the simple scheduler could be invoked [A2]. The meta-level controller could also determine 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 [A6]. The meta-level controller can hence choose to spend more resources to make a better informed decision. After getting the additional state information, the meta-level control will choose from one of the five possible choices described earlier (A7-A11).[4]



New task arrives
Drop task [A1]
Use simple scheduler on new task [A2]
Use detailed scheduler on new task [A3]
Use detailed scheduler on all tasks including partially executed tasks [A4]
Add new task to agenda [A5]
Get new features [A6]
Drop task [A7]
Use simple scheduler on new task [A8]
Use detailed scheduler on new task [A9]
Use detailed scheduler on all tasks including partially executed tasks [A10]
Add task to agenda [A11]
Legend
state
executable action
external event
transition function

Figure 3: Decision tree when a new task arrives

To elucidate this control process, instances of the state of agent *R1* and the corresponding decision choice made by the meta-level controller are provided. These are hand-generated rules specifying the required type of meta-level control decisions. These will be represented as rules in the heuristic strategies and learned automatically in the reinforcement learning strategy.

An example of the above described decision process occurs when the *R1* is in State *S1*. It represents the situation at time 2. *R1* is in a wait state doing nothing when a new task *Analyze Rock*, which arrives at time 1 with a deadline of 40, is added to the NewTaskList. *R1's* meta-level controller is invoked. All the other lists are empty and *R1* has not executed any task and has accrued zero utility. Based on its current state, *R1's* meta-level control decision is to *Call the Detailed Scheduler*.

State S1:
CurrentTime : 2

[4]The cost of computing complex features for the experiments described in this work is assumed to be low when compared to the cost of scheduling actions. This was done to test the effectiveness of these features on all the decision choices that succeed them. The cost of the computing complex features can be significantly higher than the cost of other control actions in certain domains. In those domains, it might be appropriate to reduce the number of options available after the information gathering action. For instance, if the cost of simple scheduling is 2 units and the cost of computing complex features is 4 units, it might not be sensible to do simple scheduling after computing complex features. The resources invested in computing the the complex features make only the detailed scheduling option worthwhile and not the simple scheduling option, if the choice is to schedule the task.

NewTaskList : AnalyzeRock$< 1, 40 >$; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : ϕ
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 0.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 0.0
Meta-Level Control Decision : **Call Detailed Scheduler**

Here is another instance of the meta-level control decision process where *R1* is in state *S5* and it is time 16. A new task *Explore Terrain* arrives at time 15 with a deadline of 80. The new task is added to the NewTask List and *R1's* meta-level controller is invoked. *R1* is in the midst of executing method *Focus Spectrometer on Rock*, which has executed for 2 time units. The current schedule has gained 6.0 utility points and *R1* has gained a total of 6.0 utility points also. Based on its current state, *R1's* meta-level control decision is to *Delay Explore Terrain task* and to add it to the Agenda List instead. *R1* will continue execution of method *Focus Spectrometer on Rock*. When execution of this method is completed and if the NewTask List is empty, *R1* will automatically make meta-level control decision on all the tasks in the Agenda List.

State S5:
CurrentTime :16
NewTaskList : $ExploreTerrain < 15, 80 >$; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : $\{FocusSpectrometeronRock^{exe}\}$
InformationGathered : ϕ
Utility of current schedule : 6.0; Duration of current schedule : 8.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 6.0
MLC Decision : **Add New task to agenda**

Invocation of the detailed scheduler: The parameters to the planner/scheduler are scheduling effort (E) and slack amount (S). They are determined based on the current state of the agent including characteristics of the existing schedule and the set of new tasks that are being scheduled. The *effort* parameter determines the amount of computational effort that should be invested by the planner/scheduler. The parameter can be set to either *HIGH*, where a high number of alternative plans/schedules are produced and examined or *LOW*, where pruning occurs at a very early stage and hence few alternative plans/schedules are compared, reducing the computational effort while compromising the optimality of the schedule. The effort is proportional to the expected utility and complexity (in terms of number of possible alternative plans) of the task. Although, the effort can be any discrete value, two qualitative values are used in the current implementation of the agent. These two values were sufficient to show the importance of varying the effort based on problem solving context. Depending on the problem domain, one could increase and decrease the number of feature values and the decision process will handle them appropriately.

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 three factors, the amount of uncertainty in the schedule, the importance of the currently scheduled tasks and the expected amount of 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. Three slack values of 10%, 30% and 50% of the total available time are used in the current implementation of the agent. These values, like in the case of the effort, can be varied as needed.

The decision tree describing the various action choices for this meta-level decision is shown in Figure 4. Each of the choices in the decision tree are combinations of possible effort and slack values.

An example of this type of decision process occurs when *R1* is in state *S2* and the time is 3. The new task *Explore Terrain* is to be scheduled using the detailed scheduler. *R1's* meta-level controller is invoked. Since there are no other tasks to be considered, the meta-level controller makes the following decision about parameters: EffortLevel= 2 meaning the scheduler effort should be set to HIGH and Slack of 10%, which means 10% of the total time allowed for the task in the schedule will be used for slack.

State S2:
CurrentTime : 3
NewTaskList : ϕ ; AgendaList : ϕ

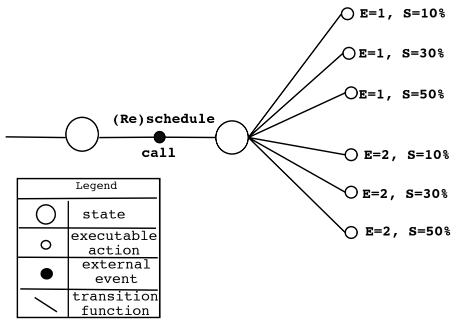


Figure 4: Decision tree for invoking the scheduler

ScheduleList: *AnalyzeRock* $< 1, 40 >$; ExecutionList : ϕ
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 0.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 0.0
MLC Decision : **Parameters for Detailed Scheduler are EffortLevel=2; Slack=10%**

Presence of task requiring coordination in current task set: Suppose there is a subtask or method in the currently scheduled task set which either requires a non-local method to enable it or should be sub-contracted out to another agent. The local agent has to decide whether it is worthwhile to even initiate negotiation and if so, which negotiation protocol to use. The decision tree associated with this meta-level decision is described in Figure 5. This decision is made using the *MetaNeg* information described below.

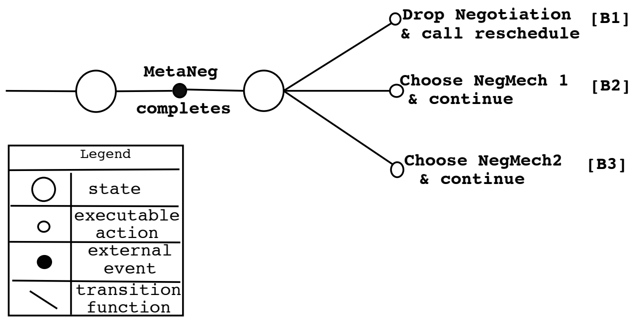


Figure 5: Decision tree on whether to negotiate and effort

Coordination actions are split into an external information gathering phase and a negotiating phase, with the outcome of the former enabling the latter. The negotiation phase can be achieved by choosing from a family of negotiation protocols [23]. The information gathering phase facilitates the negotiation phase and is modeled as a **MetaNeg** method in the task structure (see Figure 19) and the negotiation methods are modeled as individual primitive actions. Thus, reasoning about the costs of negotiation is done explicitly, just as it is done for regular domain-level activities.

The **MetaNeg** method belongs to a special class of domain actions which request an external agent for a certain set of information that does not require any significant use of local processor time. It queries the other agent and returns information such as expected utility of other agent's schedule, expected finish time of other agent's schedule, and amount of slack in the other agent's schedule. This information assists the meta-level controller in its decision making process.

The following is an instance where *R1* is in a state named *S11* at time 33. The Information Gathering Action (MetaNeg) has completed execution. The following information is returned by the information gathering action: Agent *R2* is executing high utility tasks, has deadlines which are far off and has a high amount of slack. Based on this information. *R1*'s meta-level controller is invoked and it uses the above information to decide that *R1* should negotiate with *R2* using the NegMech2 protocol about the finish time of *R2's* method *Arrive at Location.*

State S11:
CurrentTime :33

NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{NegMech2, ExamineTerrain, CollectSamples\}$
InformationGathered : $< HIGH, HIGH, HIGH$
Utility of current schedule : 0.0; Duration of current schedule : 1.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 18.0
MLC Decision : **Choose NegMech2 and continue**

Domain action completes execution: When a primitive action is completed, the meta-level controller 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. Hansen's work [13] on meta-level control of anytime algorithms using a non-myopic stopping rule is described in Section 8. It finds an intermediate strategy between continuous monitoring and not monitoring at all. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. Thus, the decision to reschedule in this paper can be viewed as a non-myopic stopping rule within Hansen's work. The decision tree associated with this meta-level decision is described in Figure 6.

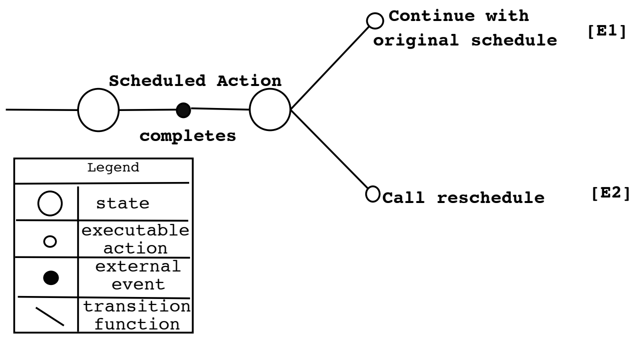


Figure 6: Decision tree when a domain action completes execution

The following is an instance where *R1* is in state *S4* and the time is 13. The new task *Analyze Rock* has been scheduled. The first action in the schedule *Get To Rock Location* has been completed successfully with a utility of is 6.0. *R1*'s meta-level controller is invoked to do a quick check to find out if the execution characteristics of this action are as expected. Since the performance is as expected, the meta-level controller decides the schedule can continue execution and the next action on the schedule *FocusSpectrometeronRock* begins execution.

State S4:
CurrentTime :13
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : $\{FocusSpectrometeronRock\}$
InformationGathered : ϕ
Utility of current schedule : 6.0; Duration of current schedule : 8.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 6.0
MLC Decision : **No Reschedule; Begin Execution of FocusSpectrometeronRock**

Negotiation process fails to reach a commitment: Suppose there is a subtask or method in the currently scheduled task set which has been negotiated about with a non-local agent and suppose the negotiation fails. The local agent should decide whether to renegotiate and if so, which protocol should it use. Figure 7 describes the associated decision tree.

The following is an instance where *R1* is in state *S15*, the time is 46 and method *NegMech2* has completed execution. *R1's* meta-level controller is invoked. The results of the negotiation is that agent *R2* will complete its method *Arrive at Location* at time 65 and this means *R1's* method *Collect Samples* can begin execution at this time. The earliest start time for this method is noted as such and the meta-level controller determines that execution of the current schedule can continue without changes.

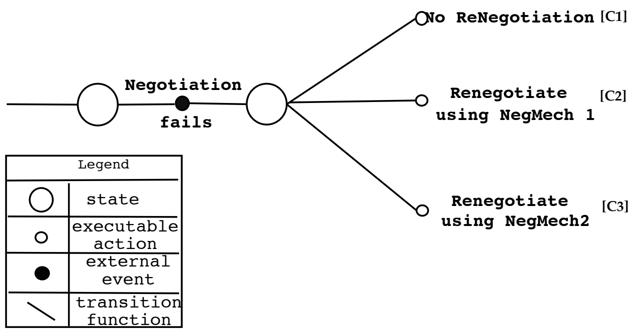


Figure 7: Decision tree on whether to renegotiate upon failure of previous negotiation

State S15:
CurrentTime :46
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ;
ExecutionList :$\{ExamineTerrain^{exe}, CollectSamples,$
$GettoRockLocation, FocusSpectrometeronRock\}$
InformationGathered : ϕ
Utility of current schedule : 1.0; Duration of current schedule : 7.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 3.0;
Total Utility accrued : 19.0
MLC Decision : **NegMech2 completes with a commitment that method CollectSamples will be enabled at time 65. Continue execution of** *ExamineTerrain*

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 Comparison to Russell and Wefald's work on Bounded Rationality

Russell and Wefald [34] introduced the idea of limited rationality and its consequences on decision making in their highly referenced work. We will now delve into a detailed description of their work and compare it to ours to situate our work within the context of the state of the art in meta-level control research. Comparisons to other related work are discussed in Section 8.

Russell and Wefald define meta-level control as the ability of an agent to choose between continued deliberation (computation) to find the action with highest expected utility and executing the current action with highest expected utility. They also show that the agent will continue to deliberate only if it is possible that the computation will change the agent's current choice of action. They describe an ideal control algorithm as one that will continue to perform the computation with the highest expected net value until no computation has positive expected value. When there is no computation left, the external action that is preferred according to the internal state resulting from the last computation is executed.

The meta-level control problem now becomes one of calculating the expected values of various computations. The advantages of doing the expected value computation are that they will help select among computations, prune pointless branches in the search tree, and terminate deliberation in such a way as to maximize overall utility of the agent. Ideally the expected value of all immediate computations should be available without making any assumptions. But these immediate computations could be the beginning of an arbitrarily long sequence of computations with multitude of potential paths making the construction of an algorithm that computes the expected value infeasible. So they approximate the expected value computation using simplifying assumptions. Particularly, they use the agent's own utility estimation function to estimate the expected value of computations.

Our definition of the meta-level control problem is similar to theirs in that we are deciding among multiple control actions that in turn affect the choice of domain actions. Our problem is slightly more complex than their problem in that they are choosing between an external action and sequence of computational actions in a single episode. We are however reasoning about and executing sequences of such sequences in other words there are multiple external actions and multiple sequences of computations that can occur in a single episode. Our Markov Decision Process-based model is equivalent to their expected utility model. In either case, the goal is to choose the sequence of control and domain(external) actions that would maximize performance in the long run.

From their model, it is clear that the knowledge necessary to assign values to computations resides in the probability distribution for the future utility estimates of the top-level actions (external actions). They assume that these probability distributions can be obtained by gathering statistics on past computations.

They postulate that computation can be characterized by some pre-defined set of features that appropriately characterize the current state of the computation. These features can be used to calculate the expected net value of taking an external action based on the current state. Their idea is that a low overhead function for mapping an action to an accurate characterization of its expected net value can be constructed using offline learning. This function can then be used online to make meta-level control decisions. An example of a simplified version of the function that maps computational time into solution quality is called a performance profile [18]. They further mention that one of the drawbacks of this approach is that if too many features are used to characterize the current state, the amount of off-line training to construct a reasonable function is sufficiently time-consuming to make the approach infeasible.

So to make the data collection process feasible, they make some simplifying assumptions in the form of myopic policies [33] described below. They claim that these assumptions can be validated only by consideration of the domain of application.

In our case, we deal with exactly the kind of complicated state that Russell and Wefald would like to avoid. The state is complex with eleven interdependent features that affect the decision process. The myopic assumptions that can be made in the Russell and Wefald work are invalid in our case for this reason. We too discuss the difficulty in obtaining enough data to obtain exact values of the probability transitions and reward function. Our solution to the problem is that we estimate the transition probabilities and reward functions using an empirical reinforcement learning approach. Also, our approach is minimally domain-dependent. The state features are an abstraction of the real state and capture information at the level of expected utility, expected duration and expected costs. Such measures allow the our meta-level control reasoning technique to be generalized to similar meta-level control problems.

Russell and Wefald define two types of computations: partial and complete computations. A partial computation does not result in commitment to an external action; a complete computation results in a commitment to an external action. A partial computation brings about changes in the internal state that will affect the value of possible further computational actions. The following are some myopic assumptions they make:

If analysis of complete computations is intractable, then consider all single primitive steps and estimate their ultimate effect; then choose the step appearing to have the highest immediate benefit. This is called the *meta-greedy algorithm.* A better approximation is to consider some but not all finite sequences of computation steps that could provide a tractable approximation to a complete computation.

However, even if only a limited set of computations is analyzed, it may be difficult to assess all the ways in which a computation can increase utility. A simplifying assumption called the *single-step assumption* would be to assume that a computation's value as a complete computation is a useful approximation to its true value as a possibly partial computation. i.e. act as though there is time for one more complete computation step. This could cause underestimation of some computations. They specifically point out that meta-greedy algorithm and single step assumption have to be jointly employed to get the advantage of the approximation. This is because if one of the two assumptions is relaxed, the other assumption gets relaxed automatically.

We too make a simplifying assumption, in that we assume that the abstract state indeed captures the critical features of the real system state. However, we validate this assumption by testing the performance of hand-generated meta-level control policies that use these features. By using these features to define the state set in our MDP, we retain the sequential end-to-end decision nature of our problem and still get a feasible solution.

In the case of partial computations where the value of current computation is defined in terms of future computation, they define an *adaptive assumption* where the current agent's choice between action and computation is correct. This will allow for a justifiable numerical estimate for the value of completed computations and theses values can be backed up allowing for the values of partial computations to be determined.

A more realistic assumption than the adaptive approach is *probabilistic self modeling* where the agents are assumed to have certain probability of taking a certain action. They define the following equation

$$U([S_i]) = \Sigma_i p(A_i) U([A_i] | A_i \ chosen)$$

This shows that an actions estimated utility is affected by the fact that it is chosen and also depends on the time at its taken and hence on the computation that ends in its beings recommended. They mention that they believe that it is possible in practice to estimate the various probabilities and conditional expected utilities involved in the equation, although they have not attempted it themselves.

Our approach can be viewed as a complete implementation of the probability estimation model described above. We show that using an estimated model indeed does allow for effective meta-level control. In this context, we can state that Russell and Wefald's work provides a solid foundation to our work and our approach complements their research.

5 Agent State

The meta-level controller uses the current state of the agent to make appropriate decisions. In this work, a distinction is made between the current state of the agent (also called real state) and the abstract representation of the state which captures only the critical information about the current state.

The real state of the agent has also the detailed information related to the agent's decision making and execution. It accounts for every task which has to be reasoned about by the agent, the execution characteristics of each of these tasks, and information about the environment such as types of tasks arriving at the agent, frequency of arrival of tasks and the deadline tightness of each of these tasks. The real state is continuous and complex. This leads to a combinatorial explosion in the real state space even for simple scenarios. The complexity of the real state is addressed by defining a abstract representation of the state which captures the important qualitative state information relevant to the meta-level control decision making process. There are eleven features in the abstract representation of the state and each feature can have one of four different values. So the maximum size of the search space is $4^{11} = 2^{22}$, which is about a million states. Framing this problem in the MDP framework would result in a search space of million states out of which only states in the order of a few thousand are actually encountered because the feature values are not independent of each other and act as constraints on each other. For instance when the utility goodness of the current schedule is HIGH and the deadline tightness of the current schedule is TIGHT, the amount of slack in local schedule is usually LOW. It never takes on the value HIGH.

The following are some characteristics of system state features and the environment.

1. The status of tasks currently being processed and those which need future processing. Example: NewTaskList, AgendaList, ScheduleList, ExecutionList
2. Objective function Example: Maximize utility of tasks, Minimize schedule durations and cost of tasks
3. Environmental model Example: Probability of arrival of tasks, task complexity and their deadline tightness
4. Internal influences on action choice Example: Slack in the schedules
5. External influences on action choice Example: Utility of tasks of other agents, Deadline Tightness of tasks belonging to other agents, Slack in schedules of other agents
6. Real time performance characteristics Example: Deviation from expected performance, Cost of decommiting from existing tasks

These "real" features are used to construct the abstract state representation that will permit effective meta-level control for the domain described in this paper.

5.1 Abstract Representation of the State

The overhead of meta-level control activities is accounted for by the cost of state feature computation. The eleven features, which are of two categories - simple features where the reference values are readily available by simple lookups and complex features which involve significant amount of computation to determine their values.

Simple features help the agent make informed decisions on executable actions or whether to obtain more complex features to make the decisions. An example of a simple feature would be the availability of slack in the current schedule. If there is a lot of slack or too little slack, the decision to accept the new task or drop the new task respectively is made. However, if there is a moderate amount of slack, the agent might choose to obtain a more complex feature, namely computing the relation of slack fragments which is described below.

Complex features usually involve computations that take time that is sufficiently long that, if not accounted for, will lead to incorrect meta-level decisions. The computation of the complex features is cumbersome since they involve determining detailed timing, placement and priority [5] characteristics and provide the meta-level controller with information

[5]Priority accounts for quality and deadline.

FeatureID	Feature	Complexity
F1	Utility goodness of new task	Simple
F2	Deadline tightness of a new task	Simple
F3	Utility goodness of current schedule	Simple
F4	Deadline tightness of current schedule	Simple
F5	Arrival of a valuable new task	Simple
F6	Amount of slack in local schedule	Simple
F7	Amount of slack in other agent's schedule	Simple
F8	Deviation from expected performance	Simple
F9	Decommitment Cost for a task	Complex
F10	Relation of slack fragments in local schedule to new task	Complex
F11	Relation of slack fragments in non-local agent to new task	Complex

Table 1: Table of proposed state features, their description and category

to make more accurate action choices. For instance, instead of having a feature which gives a general description of the slack distribution in the current schedule i.e. there is a lot of slack in the beginning or end of the schedule, there is a feature which examines the exact characteristics of the new task and makes a determination whether the available slack distribution will likely allow for a new task to be included in the schedule. The agents make explicit meta-level control decisions based on whether to gather complex features and determine which complex features are appropriate.

In Section 2, a formal definition of the meta-level control problem was presented. The abstract representation of the state defined in this section will be the states in the Markov Decision process model. The control actions defined in section 3 will the actions in the MDP model. The probability transition function and the reward function will be determined by estimating them from data gathered in previous system runs.

Table 1 enumerates the features of the abstract representation of the state used by the meta-level controller. The default value or each of the features is NONE.

F1: Utility goodness of new task: It is a simple feature which 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. The assigned feature values are HIGH, MEDIUM and LOW respectively.

F2: Deadline tightness of a new task: It is a simple feature which 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. The assigned feature values are TIGHT, MEDIUM, LOOSE respectively.

F3: Utility goodness of current schedule: It is a simple feature 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. The assigned feature values are HIGH, MEDIUM and LOW respectively.

F4: Deadline tightness of current schedule: It is a simple feature which describes the deadline tightness of the current schedule in relation to expected deadlines of tasks in that environment. If there are multiple tasks with varying deadlines in the schedule, the average tightness of their deadlines is computed. It determines whether the schedule's deadline is very close, moderately close or far in the future.The assigned feature values are TIGHT, MEDIUM, LOOSE respectively.

F5: Arrival of a valuable new task: It is a simple feature which 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. It can take on the values of HIGH, MEDIUM, LOW.

F6: Amount of slack in local schedule: It is a simple feature which provides a quick evaluation of the flexibility in the local schedule. Availability of slack means the agent can deal with unanticipated events easily without doing a reschedule. The cost of inserting slack is that the available time in the schedule is not all being used to execute domain actions. This feature can take on the values of HIGH, MEDIUM, LOW.

F7: Amount of slack in other agent's schedule: This is a simple feature used to make a quick evaluation of the flexibility in the other agent's schedule. This is used when an agent is considering coordinating with the other agent to complete a task. This feature can take on the values of HIGH, MEDIUM, LOW.

F8: Deviation from expected performance: This is a simple feature which uses expected performance characteristics of the schedule and the current amount of slack (F6) to determine by how much actual performance is deviating from expected performance. The feature can take on the values of HIGH, MEDIUM, LOW.

F9: Decommitment Cost for a task: This is a complex feature which estimates the cost of decommiting from doing a method/task by considering the local and non-local down-stream effects of such a decommit. This feature can take on the values of HIGH, MEDIUM, LOW.

F10: Relation of slack fragments in local schedule to new task: This is a complex feature which determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular schedule. It involves resolving detailed timing and placement issues. This feature can take on the values of HIGH, MEDIUM, LOW.

F11: Relation of slack fragments in non-local agent to new task: This is a complex feature which determines the feasibility of fitting a new task given the detailed fragmentation of slack in a particular non-local schedule. This feature can take on the values of HIGH, MEDIUM, LOW.

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 later sections, these qualitative measures provide information that can be exploited to make effective meta-level control decisions.

5.2 Computation of State Features

As described above, our goal is to bound the complexity of the state space. This is done by making the high level state features time independent and also by eliminating the specifics of tasks and their performance characteristics from the state. We adopt a qualitative representation for the state feature values and consider the characteristics of task sets instead of individual tasks.

The following describes the formal mechanism that exploits knowledge about the agent tasks and environmental characteristics to determine the high-level features of the agent state.

Definition 1: *The multi-agent system M is a collection of n heterogeneous agents. Each agent α has a finite set of tasks T which arrive in a finite interval of time. N_Ψ is the total number of tasks that have arrived at the system from the start to current time Ψ. Let $t \epsilon T$ be a single task under consideration*

Definition 2: *A task t upon arrival has an arrival time AT_t and a deadline DL_t associated with it. A task t can be achieved by one of various alternative ways(plans) $t^1, t^2, t^3 ... t^k$.*

Definition 3: *A plan t^j to achieve task t is a sequence of executable primitive actions $t^j = \{m_1, m_2, ...m_n\}$. Each plan t^j has an associated utility distribution UD_{t^j} and duration distribution DD_{t^j}.*

Example: $ExamineTerrain^A$ and $ExamineTerrain^B$ are two alternate plans to achieve task *ExamineTerrain*. (25% 22 50% 50 25% 100) is the duration distribution of $ExamineTerrain^A$, which means that plan $ExamineTerrain^A$ takes 22 units of time 25% of the time, 50 time units 50% of the time and 100 time units 25% of the time. Also $ExamineTerrain^A$ has a utility distribution of (10% 30 90% 45). $ExamineTerrain^B$ has a duration distribution (50% 32 30% 40 20% 45) and utility distribution of (25% 20 75% 30).

$$\begin{aligned} UD_{ExamineTerrain^A} &= (10\%\ 30\ 90\%\ 45) \\ DD_{ExamineTerrain^A} &= (25\%\ 22\ 50\%\ 50\ 25\%\ 100) \\ UD_{ExamineTerrain^B} &= (25\%\ 20\ 75\%\ 30) \\ DD_{ExamineTerrain^B} &= (50\%\ 32\ 30\%\ 40\ 20\%\ 45) \end{aligned}$$

Definition 4: *Ψ_t is the time required for scheduling a task t if it is chosen for scheduling.*

Example : Ψ_t is 2 units if simple scheduling is chosen, if detailed scheduling is chosen, the cost is 4 units if the scheduling set has less than 5 primitive actions to evaluate, 12 units if the scheduling set has between 5 and 10 primitive actions to evaluate and 18 units if the scheduling set has more than 10 primitive actions.

System execution is single threaded allowing for one primitive action at the most to be in execution at any time. If a meta-level action is required when a primitive action m is executing, the execution is interrupted and control is turned over to the meta-level controller. When the meta-level control action is completed, execution of m is always resumed.[6]

[6]This is an artifact of the simulation environment. In order for the simulator to keep accurate records of utility accumulated, it requires that executing actions should be fully completed.

Definition 5: *Υ_m is the remaining time required for primitive action m to complete execution.*

Definition 6: *The earliest start time EST_t for a task t is the arrival time AT_t of the task delayed by the sum of Υ_m, the time required for completing the execution of the action m which is interrupted by a meta-level control event and Ψ_t, the time required for scheduling the new task.*

$$EST_t = AT_t + \Upsilon_m + \Psi_t$$

Definition 7: *The maximum available duration MD_t for a task t is the difference between the deadline of the task and its earliest start time.*

$$MD_t = DL_t - EST_t$$

Example: Suppose *ExamineTerrain* arrives at time 45 and has a deadline of 100. Also suppose the execution of method m is interrupted by the arrival of *ExamineTerrain* and m still needs about 8 time units to complete execution. Suppose the time spent on scheduling *ExamineTerrain* is 5 units. Then the maximum available duration for task *ExamineTerrain* is $100 - 45 - 8 - 5 = 42$ time units. The meta-level controller is aware that the entire range of the maximum available duration is not always available solely for the execution of this task. When the maximum available duration ranges of a number of tasks overlap, the maximum duration available for a particular task is effectively reduced.

Definition 8: *Given a task t and its maximum available duration MD_t, the probability that a plan t^j meets its deadline PDL_{t^j} is the sum of the probabilities of all values in the duration distribution of plan t^j which are less than the task's maximum available duration.*

$$PDL_{t^j} = \sum_{j=1}^{n} \frac{p_j}{100} : ((p_j\% \; x_j) \in DD_{t^j}) \wedge (x_j < MD_t)$$

Example: Suppose the maximum available duration for task *ExamineTerrain* is 42. There is only one duration value in $DD_{ExamineTerrain^A}$ which has a value less than 42 and that value is 22 and occurs 25% of the time.

$$PDL_{ExamineTerrain^A} = \frac{25}{100} = 0.25$$

There are two duration values in $DD_{ExamineTerrain^B}$ which have a value less than 42 and they are 32 and 40 which occur 50% and 30% respectively in the distribution.

$$PDL_{ExamineTerrain^B} = \frac{50 \; + \; 30}{100} = 0.8$$

Definition 9: *The expected duration ED_{t^j} of a plan t^j, is the expected duration of all values in the duration distribution of plan t^j which are less than the maximum available duration for the task.*

$$ED_{t^j} = \frac{\sum_{j=1}^{n} \frac{p_j}{100} * x_j}{PDL_{t^j}} : ((p_j\% \; x_j) \in DD_{t^j}) \wedge (x_j < MD_t)$$

Example: For the above constraint where the maximum available duration for task *ExamineTerrain* is 42

$$ED_{ExamineTerrain^A} = \frac{(\frac{25}{100} * 22 \;)}{0.25} = 22$$

$$ED_{ExamineTerrain^B} = \frac{(\frac{50}{100} * 32 \; + \; \frac{30}{100} * 40)}{0.8} = 35$$

Definition 10: *The expected utility EU_{t^j} of a plan t^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 t^j.*

$$EU_{t^j} = \sum_{j=1}^{n} PDL_{t^j} * \frac{p_j}{100} * x_j : ((p_j\% \; x_j) \in UD_{t^j})$$

Example: When the maximum available duration for task *ExamineTerrain* is 42,

$$EU_{ExamineTerrain^A} = 0.25 * \frac{10}{100} * 30 \; + \; 0.25 * \frac{90}{100} * 45 = 10.875$$

$$EU_{ExamineTerrain^B} = 0.8 * \frac{25}{100} * 20 \; + \; 0.8 * \frac{75}{100} * 30 = 22$$

Definition 11: *Given the maximum available duration for a task, the preferred alternative ALT_t for a task t is the alternative whose expected utility to expected duration ratio is the highest. ALT_t is the alternative which has the potential to obtain the maximum utility in minimum duration within the given deadline.*

$$ALT_t = t^j : \max_{j=1}^{n} \frac{EU_{tj}}{ED_{tj}}$$

Example: Suppose the maximum available duration for task *ExamineTerrain* is 42. Consider each of *ExamineTerrain*'s alternative plans which were described earlier. Plan $ExamineTerrain^A$'s expected utility to expected duration ratio is $\frac{10.875}{22} = 0.494$ and plan $ExamineTerrain^B$'s expected utility to expected duration ratio is $\frac{22}{35} = 0.629$. So the alternative with the maximum expected utility to expected duration ratio[7] is $ExamineTerrain^B$.

$$ALT_{ExamineTerrain} = ExamineTerrain^B$$

. **Definition 12:** *The utility goodness UD_t of a task t is the measure which determines how good the expected utility to expected duration ratio of a 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 tasks which are in the 66th percentile(top 1/3rd) of the expected utility to expected duration ratio of the task's preferred alternative.

$$UD_t = \begin{cases} HIGH, \; \frac{EU_{ALT_t}}{ED_{ALT_t}} \; is \; above \; the \; 66th \; percentile \\ MEDIUM, \; \frac{EU_{ALT_t}}{ED_{ALT_t}} \; is \; between \; the \; 66th \; and \; 33rd \; percentile \\ LOW, otherwise \end{cases}$$

Example: The utility goodness of task *ExamineTerrain* given a deadline of 100 and a maximum available duration of 42 is $\frac{22}{35} = 0.628$ which lies above the 66th percentile. $UD_{ExamineTerrain} = HIGH$

Definition 13: *The deadline tightness TD_t of a task t measures the flexibility of the maximum available duration. It determines by how much the maximum available duration can be reduced by unexpected meta-level activities and similar delays and the system can still guarantee the same performance characteristics of the preferred alternative for the task.* Suppose a meta-level activity on average has an expected duration of C_{ML}. The *expected amount of time required for handling unexpected meta-level activities* Ω_t, during the execution of task t, is computed as follows:

$$\Omega_t = C_{ML} * \frac{N_\Psi}{\Psi} * MD_t$$

Example: Suppose the average time per meta-level activity is 2 units, 4 tasks have arrived at the agent and the current time is 60. MD_t is 42 as determined previously. $\Omega_{ExamineTerrain} = 2 * \frac{4}{60} * 42 = 5.6$ The amount of time expected to be spent on future meta-level activities is 5.6 units.

In order to determine if a given deadline is tight, the *proposed maximum available duration*, MD_t^X for a proposed scenario X is computed. It is the maximum available duration which also accounts for the anticipated meta-level costs of future activities.

$$MD_t^X = MD_t - \Omega_t$$

Example: From the previous example, the $MD_{ExamineTerrain}^X = 42 - 5.6 = 36.4$

The related parameters $PDL_{ALT_t}^X$, $ED_{ALT_t}^X$, $EU_{ALT_t}^X$ and the expected utility to expected duration ratio $\frac{EU_{ALT_t}^X}{ED_{ALT_t}^X}$ for the proposed scenario are also recomputed with respect to the redefined MD_t^X.

Example: Following through with the example described above, the new parameter values are:

$$PDL_{ExamineTerrain^B}^X = 0.5, \quad ED_{ExamineTerrain^B}^X = 32$$

$$EU_{ExamineTerrain^B}^X = 11.25, \quad UD_{ExamineTerrain}^X = \frac{EU_{ExamineTerrain^B}^X}{ED_{ExamineTerrain^B}^X} = 0.3451$$

The expected utility to expected duration ratio now falls below the 33rd percentile,

$$UD_{ExamineTerrain}^X = LOW$$

[7]The assumption here is that there may be other tasks that can use the available time. If we add a model of opportunity cost, this definition can be modified. This is an area of future work.

$$TD_t = \begin{cases} TIGHT, (UD_t \ = HIGH) \wedge (UD_t^X \neq \ HIGH \\ LOOSE, (UD_t \ = HIGH) \wedge (UD_t^X = HIGH) \\ MEDIUM, \forall \ other \ values \ of \ UD_t, \ UD_t^X \end{cases}$$

Example: Since $(UD_{ExamineTerrain} = HIGH) \wedge (UD_{ExamineTerrain}^X = LOW)$, the time spent on unexpected meta-level control activities is detrimental to task t's utility gain, which in turn means its deadline is tight.

$$TD_{ExamineTerrain} = TIGHT$$

Definition 14: *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.*

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

Example:

$$HPTS_A = \{ExamineTerrain\}$$

Definition 15: *The arrival rate of high priority tasks 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.*

$$ART_\alpha = \frac{|T_k|}{n} : T_k \ \epsilon \ HPTS_\alpha$$

Definition 16: *The probability of a high priority task arriving in the near future PHT_α depends on the arrival rate of high priority tasks. The intuition behind this relation is that the characteristics of tasks that arrived in the past can be used to predict the characteristics of tasks that will arrive in the near future. The assumption made by the system that the past information can be used to predict the future is a valid assumption since the environment is stationary for a finite-horizon.*

For instance, if ART_α is less than 0.04 (arrival rate is less than 4%), then PHT_α is also low.

$$PHT_\alpha = \begin{cases} LOW, \quad ART_\alpha < 0.04 \\ MEDIUM, \quad 0.04 <= ART_\alpha < 0.10 \\ HIGH, \quad ART_\alpha >= 0.10 \end{cases}$$

Definition 17: *The slack in the schedule $SLACK_{scur}$, is the total amount of flexibility that should be inserted in the schedule so that unexpected meta-level activities and uncertainty in method execution durations of all the tasks being scheduled can be accommodated without expensive rescheduling control actions.*

The cost of unexpected meta-level activities is Ω_t which was previously defined. The uncertainty in method durations is handled by the complex scheduler which reasons about uncertainty.

$$SLACK_{scur} = \sum_{\forall t \epsilon scur} \Omega_t$$

The slack is defined using a simple slack distribution strategy, where the duration of each method in the schedule is extended by equal fractions of the total slack.

$SLACK_{scur}^t$ is the slack remaining in the midst of a schedule *scur*'s execution at time t.

Definition 18: *The expected utility of the current schedule scur (EU_{scur}) is provided by the domain scheduler when it completes constructing a schedule. The information on the expected duration of the schedule ED_{scur}, which is the sum of the execution durations of the primitive actions in the plan, is also provided by the scheduler. The expected start time of the schedule EST_{scur} and expected finishing time of the schedule EFT_{scur} are also provided by the domain scheduler.*

The following is an example of a meta-level decision making scenario occurring in the system.

Scenario: Suppose a new task t arrives when the agent α is in the midst of executing actions from the current schedule *scur*.

Meta Level Action: If there is a high probability that the scheduler will not select any plan to execute the new task, then the meta-level control will choose to **drop the task** even before it is sent to the scheduler to avoid the cost of scheduling. The scheduler will not create a plan for executing a task t if it determines that the utility obtained from excluding the task is higher than the utility obtained by including the task in the schedule.

Heuristic 1: The *probability of the scheduler not selecting* any of the alternative plans for a new task for execution can be high for one or more of the following reasons:

1. There is no alternative for the task whose expected duration is less than the task's maximum available duration. This means the deadline is so tight that there is no way to complete the task within the deadline.

$$\forall j, PDL_{tj} = 0.0$$

2. The expected utility of the current schedule is much higher than the expected utility of the new task. Also the expected deadline of the current schedule is tight enough that any alterations in the schedule could result in the expected quality to be seriously depleted due to missed deadlines and broken commitments,

$$(TD_t = TIGHT) \wedge (\frac{EU_{scur}}{ED_{scur}} \gg \frac{EU_t}{ED_t}) \wedge (ED_t \gg SLACK_{scur})$$

Heuristic 2: Samuelson [35], states that the opportunity cost of a decision arises because choosing one thing in a world of scarcity means giving up something else. Opportunity cost(OC) is defined as *the value of the good or service foregone.* The opportunity *cost of (re)scheduling* is too high if one or more of the following occur

1. The time spent on execution of the control action(scheduling) the new task delays the execution of previously scheduled domain actions. These delays could result in broken commitments, missed deadlines and lowered utilities which contribute to the cost of the delays. If the cost of these delays are much higher than the expected benefits of the scheduling action, then the opportunity cost of rescheduling is too high.

   If Ψ is the current time and t is the new task being scheduled, then OC is too high if the scheduling event causes the task to use more slack than was allocated making the tasks in the previously existing schedule to miss their deadline. The slack allocated amount can be insufficient if primitive actions take significantly longer execution durations than expected and also if there are significantly more number of meta-level events than expected for that time interval

$$(\frac{EU_{scur}}{ED_{scur}} \gg \frac{EU_t}{ED_t}) \wedge (C_t >> SLACK^{\Psi}_{scur})$$

2. The time spent on scheduling and switching contexts from the control layer to meta-level control layer could lead to the delay of meta-level control decisions on other potentially high priority tasks which arrive during that time period. This delay could result in lowered utilities from the latter tasks. If the gain from the scheduling action is less than the loss of utility from other control actions, then the opportunity cost of the scheduling action is too high.

   Suppose, a new task arrives t and the agent α isn't executing a schedule, then the OC to schedule the new task is too high if the probability of high priority tasks arriving during the scheduling duration is significantly high.

$$(TD_t = Tight) \wedge (PHT_\alpha = HIGH)$$

This section describes the meta-level control problem in cooperative multi-agent systems. Meta-level control is the ability of complex agents operating in open environments to sequence domain and control actions to optimize expected performance. Meta-level control supports decisions on when to accept, delay or reject a new task, when it is appropriate to negotiate with another agent, whether to renegotiate when a negotiation task fails, 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. These decisions influence each other and affect the amount of resources available for future computations. A description of a meta-level agent architecture with bounded computational overhead which supports the sequential decision making process is provided. We also describe how knowledge about tasks and environmental characteristics are exploited to constrain the size of the state space. Information about time (eg. task arrival time, task achievement time) and qualitative task characteristics (eg. quality, cost, duration) are eliminated in the high-level representation of the agent state, Instead, a more generalized view of system performance is captured.

The following sections will test the hypothesis that using this state information, the best sequence of control and domain actions can be determined for each environment. The action sequence can either be determined by a heuristic hand-generated rules as described in the next section or can be learned automatically as described in Section 7.

6 Heuristic Strategies

This section addresses the three following questions: Does meta-level control lead to better performance in rational agents situated in the domain described in this paper? Is it possible to construct a hand-generated meta-level control policy based on the high-level state features described earlier for specific environments. Does this hand-generated policy outperform a deterministic meta-level control policy?

Two heuristic strategies, the Naive Heuristic Strategy and the Sophisticated Heuristic Strategy, that use context sensitive rules for meta-level control are described. Both strategies use high-level state features and they serve as a test-bed for the effectiveness of the state features for efficient meta-level control. The two strategies differ from each other in that the naive strategy does not use information on the future arrival of tasks in its reasoning process and hence makes myopic decisions. The sophisticated heuristic strategy on the other hand makes non-myopic decisions using probabilistic information about future events that is available to it.

6.1 Sophisticated Heuristic Strategy

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 work, the information on the three parameters is available to the SHS. Though not implemented in the context of this paper, this information can be learned by the SHS by gathering statistics over multiple runs. The meta-level controller can make non-myopic decisions by including information about its environment in its reasoning process.

The Naive Heuristic Strategy (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. The following are the 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.

We now describe some of the rules supporting the various action choices for three of the five meta-level control event triggers. We provide a sample of the rules instead of complete list in the interest of space.

Table 2 and Table 3 describes SHS rules required to support each of the actions on the new task in Figure 3 for the *Arrival of new task* event trigger. For instance the first row in the table describes the following rule: If new task has LOW utility goodness and TIGHT deadline; HIGH probability of high priority tasks arriving in the near future, then best action is *Drop Task (A1)*. When a feature is not specifically addressed in a rule, it is assumed that the feature can take on any of its domain values. So in the above example the Agenda Utility Goodness (AGUG) can be HIGH, MEDIUM Or LOW and the rule would still hold true.

Table 4 describes SHS rules required to support each of the actions in Figure 4 for the *Invoking domain level scheduler* event trigger.

Table 5 describes SHS rules required to support each of the actions in Figure 6 for the *Domain action completes execution* event trigger. The deviation in expected performance(DEV) given by

$$DEV = \frac{actual\ utility\ gained\ from\ action\ -\ expected\ utility\ of\ action}{expected\ utility\ of\ action}$$

6.2 Single-agent Experiments

This sub-section provides performance comparisons of four different strategies to meta-level control: Naive Heuristic Strategy (NHS); Sophisticated Heuristic Strategy (SHS); Deterministic Strategy; and Random Strategy within a single agent context. We define a deterministic strategy that 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. The deterministic strategy also does not automatically reschedule upon execution failure. The random strategy randomly chooses its actions for each of the three meta-level control decisions.

The meta-level control decisions that are considered in this single agent set up are: 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. For all the experiments the following costs are

NTUG	NTDL	CSUG	CSDL	P(HPT)	MLC Decision
L	T	*	*	H	Drop Task (A1)
L	*	H	T	*	Drop Task (A1)
H	M/T	H	M/T	H	Drop Task (A1)
H	T	L	*	*	Use Simple Scheduler (A2)
L	*	-	-	L	Use Simple Scheduler (A2)
L	*	*	*	*	Use Simple Scheduler (A2)
H	M	M	*	M	Use Detailed Scheduler (A3)
H	T	L	T	L	Use Detailed Scheduler (A3)
H	T	L	T	L	Use Detailed Scheduler on All Lists (A4)
H	LS	H	M/LS	L/M	Add New Task to Agenda (A5)
M/L	M/LS	H	*	L	Add New Task to Agenda (A5)

Table 2: SHS rules for *Arrival of New Task* event trigger (Actions A1-A5). The column headers are NTUG = New Task Utility Goodness; NTDL = New Task Deadline; CSUG = Current Schedule Utility Goodness; CSDL = Current Schedule Deadline; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

NTUG	NTDL	CSUG	CSDL	CSSL	AGUG	AGDL	DC	P(HPT)	MLC Decision
*	*	M/H	LS/M	*	M/H	LS/M	*	L	Get More Features (A6)
*	*	M	M	L	M	M	L/M	*	Drop Task (A7)
*	*	M/H	M	*	M/H	M	*	*	Use Simple Scheduler (A8)
M/H	M	M/H	M	H	*	*	*	*	Use Detailed Scheduler (A9)
H	*	H	M	H	H	*	M	*	Use Detailed Scheduler All Lists (A10)
H	LS	H	M	M	*	*	*	*	Add New Task to Agenda (A11)

Table 3: SHS rules for *Arrival of New Task* event trigger (Actions A6-A11). The column headers are NTUG = New Task Utility Goodness; NTDL = New Task Deadline; CSUG = Current Schedule Utility Goodness; CSDL = Current Schedule Deadline; CSSL = Slack in Current Schedule; AGUG = Utility Goodness of tasks in Agenda; UGDL = Deadline Tightness of tasks in Agenda; DC = Decommitment Cost; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

assumed. The meta-level control actions have an associated cost of 1 time unit; the drop task and delay task actions take 1 time unit also. The call to simple scheduler costs 2 time units and the cost of computation of complex features costs 2 time units, the cost of detailed scheduling tasks with less than five methods is 4 units, with less than ten methods is 12 time units and greater than ten methods is 18 time units. The duration of the control action is proportional to the effort level. Actions requiring higher effort levels will require longer durations.

The agents in the experimental test-bed were implemented using the Java Agent Framework (JAF) framework [49] and situated in the Multi-Agent Survivability Simulator (MASS) environment [17]. Each agent simulation was run on a Intel Pentium(R) machines with four 1.80 GHz processors running linux. Each machine has 256 MB of memory and connected via a fast-ethernet network interface.

The task environment generator randomly creates task structures while varying three critical factors:

1. complexity of tasks $c \epsilon \{simple(S), complex(C), combination(A)\}$
2. frequency of arrival $f \epsilon \{high(H), medium(M), low(L)\}$
3. tightness of deadline $dl \epsilon \{tight(T), medium(M), loose(L)\}$.

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. A simple task has two primitive actions and its structure and number of possible

NTUG	NTDL	NTUNC	CSUG	CSDL	P(HPT)	MLC Decision
M/L	M	*	M	M/LS	L	EffortLevel = 1
H	T/LS	*	H/L	T	H/M	EffortLevel = 2
*	*	H	*	*	M	Slack = 50%
*	*	M	*	*	H/M	Slack = 30%
*	*	L	*	*	L	Slack = 10%

Table 4: SHS rules for *Invoking Domain Level Scheduler* event trigger. The column headers are NTUG = New Task Utility Goodness; NTDL = New Task Deadline; NTUNC = Uncertainty in execution characteristics of new task; CSUG = Current Schedule Utility Goodness; CSDL = Current Schedule Deadline; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

Deviation	P(HPT)	MLC Decision
$< 10\%$	*	Continue with Original Schedule (E1)
$> 10\%$	H	Continue with Original Schedule (E1)
$> 10\%$	M/L	ReSchedule (E2)

Table 5: SHS rules for *Domain Action Completes Execution* event trigger. The column headers are Deviation = Deviation in expected performance; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

alternatives is similar to the AnalyzeRock task (Figure 1)described in Section 2. The utility distribution and duration distribution of a simple task is within a 5% range of the corresponding distributions of AnalyzeRock. A complex task also has structure similar to that of GetImage task described in Figure 8. It has between four and six primitive actions. The utility distribution and duration distribution of a complex task is within a 5% range of the corresponding distributions of GetImage. The combination value means that 50% of the tasks are simple and 50% are complex tasks.

The frequency of arrival of tasks refers to the number of tasks that arrive within a finite time horizon. The resource contention among the tasks increases as the task frequency increases. Task arrival is is determined by a normal distribution with $\mu = 250$ and $\sigma = 249$. When the frequency of arrival is low, about one to ten tasks arrive at the agent in 500 time unit horizon; when the frequency is medium, between ten and fifteen tasks arrive at the agent; and when the arrival frequency is high, fifteen to twenty arrive on average at the agent. 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. If the deadline tightness is set to low, the maximum available duration given to the task is between 120% and 150% of the expected duration of the task; if the deadline tightness is set to medium, the maximum available duration given to the tasks is between 100% and 120% of the expected duration of the task; and if the deadline

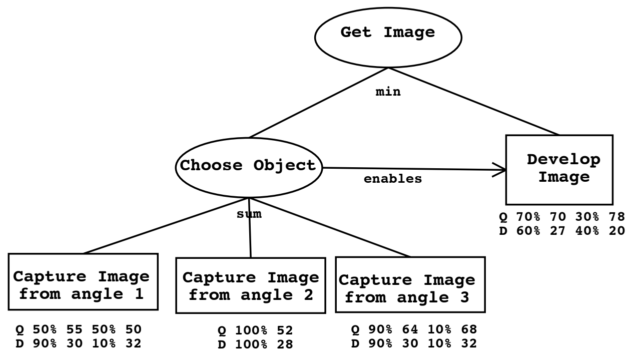


Figure 8: A Complex Task in a Single Agent Environment

Row#		SHS	NHS	Deter.	Rand.
1	AUG	205.49	192.10	121.90	89.97
2	σ	7.0	12.5	12.55	19.114
3	CT	20.37%	23.92%	39.27%	11.77%
4	RES	0%	14.53%	0%	50.56%
5	PTC	41.08%	39.64%	30.52%	21.56%
6	PTDEL	43.78%	49.0%	0%	11.49%

Table 6: Performance evaluation of four algorithms over a single environment AMM 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); Row 5 is the percent of total tasks completed (PTC);Row 6 is percent of tasks delayed on arrival (PTDEL)

tightness is set to high, the maximum available duration is between 80% and 100% of the expected duration of the task.

Environments are named based on values of these three criteria in the order mentioned above. For instance, environment AMM is one that has a combination of simple and complex tasks, with medium frequency of arrival and medium deadline tightness.

The experimental results described in Table 6 show the performance of the various strategies in the environment, AMM, which, as mentioned before, contains a combination of simple and complex tasks. The frequency of task arrival in this environment is medium and ranges between 10 and 15 tasks in the 500 time unit interval. The deadline tightness is also medium. Simple tasks have a minimum duration of 8 time units and a maximum duration of 20 time units while complex tasks take a minimum duration of 50 time units and a maximum duration of 120 time units. Also the utility gained by completing a single task can range between 6 and 24 while the utility gained from a complex task is between 70 and 80. Each strategy was evaluated over 300 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.

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 of the table describe the average utility gained (AUG) by each of the strategies and the corresponding standard deviations. The heuristic strategies (SHS and 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. Detailed analysis of the data shows 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 automatically 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).

Row 5 is the percent of total tasks completed (PTC). This was found to be less than 50% for this environment. This is because this environment is fairly dynamic (in terms of frequency of occurrence of exogenous events) and has tight constraints (the deadlines of task are of medium tightness) that limit the number of tasks that can be successfully completed

Row 6 is percent of tasks delayed on arrival (PTDEL). Here too about 45% of the tasks are delayed in case of the

Environment	SHS	NHS	Deter.	Rand.	p1	p2	p3
AMM	205.49	192.10	121.90	89.97	0.032	0.0001	0.0001
AMT	117.34	115.69	82.17	67.33	0.4391	0.0001	0.0001
AHT	124.80	123.96	61.77	86.20	0.6906	0.0001	0.0001
ALM	135.05	124.74	115.93	48.21	0.004	0.0001	0.0001
AML	231.44	218.07	140.80	105.16	0.0045	0.0001	0.0001
AHL	229.07	218.86	94.55	127.47	0.0024	0.0001	0.0001
ALL	151.31	145.03	130.80	51.76	0.2596	0.0001	0.0001
CLL	163.77	157.27	103.33	50.86	0.0643	0.0001	0.0001

Table 7: Utility Comparisons over a number of environments

heuristic strategies signifying there is significant overlap among the tasks in terms of resource usage. In other words, new tasks often arrive at the agent when the agent is busy with other tasks.

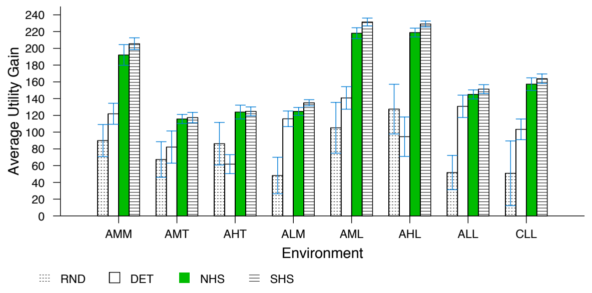


Figure 9: Average Utility Comparison between Heuristic Strategies and Baseline Strategies over 8 different environments. The error bars are one standard deviation above and below each mean

Figure 9 shows the utility comparisons over a number of environments. The heuristic strategies (SHS and NHS), as in the case of environment (AMM) described previously, significantly outperform ($p<0.05$)the baseline strategies (Deterministic and Random) over all eight types of environments. The accepted hypothesis is that SHS and NHS on average achieved at least 30% more utility than the deterministic strategy.

Table 7 provides the detailed information on the performance comparison. Columns 2-5 show the average utility gained by each of the four algorithms for that environment. Column 6 named p1 shows the statistical significance (p-value) of SHS with respect to NHS. Column 7 named p2 shows the statistical significance of SHS with respect to the deterministic algorithm. Column 8 named p3 shows the statistical significance of NHS with respect to the deterministic algorithm.

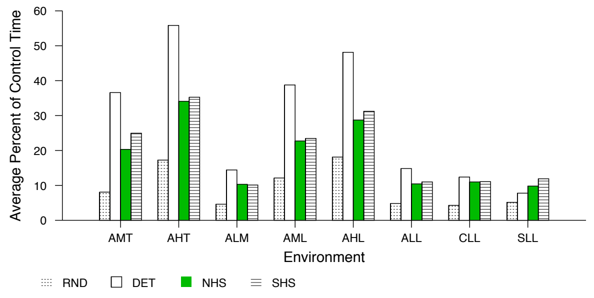


Figure 10: Average Percent Control Time Comparison between Heuristic Strategies and Baseline Strategies over 8 different environments

The reason for the improved performance by the heuristic strategies when compared to the deterministic and random

Environment#	SHS	NHS	Deter.	Rand.
AMM	20.37%	23.92%	39.27%	11.77%
AMT	24.95%	20.32%	36.59%	8.07%
AHT	35.26%	34.09%	55.82%	17.24%
ALM	10.11%	10.32%	14.42%	4.61%
AML	23.45%	22.73%	38.77%	12.12%
AHL	31.23%	28.73%	48.12%	18.11%
ALL	10.99%	10.44%	14.83%	4.82%
CLL	11.08%	10.99%	12.39%	4.29%

Table 8: 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;

strategies is found in Figure 10 which shows the percent of control time comparisons over the same set of environments. As described in earlier, control actions do not have associated utility of their own. Domain actions produce utility upon successful execution and the control actions serve as facilitators in choosing the best domain actions given the agent's state information. So resources such as time spent directly on control actions do not directly produce utility. When excessive amounts of resources are spent on control actions, the agent's utility is reduced since resources are bounded and are not available for utility producing domain actions.

The heuristic strategies use control activities that optimize their use of available resources (time in this case). The deterministic strategy on the other hand always makes the same control choice, the expensive call to the detailed scheduler, independent of context. Hence the deterministic strategy has higher control costs, than the heuristic strategies and has less resources (time) to execute domain actions and accrue utility. The random strategy has low control costs but it doesn't reason about its choices leading to bad overall performance. Table 8 provides the details about the percent of total available time per episode(500 units) that was spent on control actions.

It can be observed in Table 7 that the SHS strategy is significantly better than the NHS ($p<0.05$) in some environments (ALM, AML, AHL). All three environments can be characterized as medium constrained environments. In environment ALM, the arrival frequency is loosely constrained while the deadline tightness is MEDIUM. On detailed analysis of the data, it was found that there were extended periods in which no tasks arrived at the agent and then there would be burst of task arrivals. So information on the nature of future tasks allowed the agent to make better decisions during those periods of resource contentions (caused by the medium deadlines). In the other two environments, the deadline tightness was LOOSE while the arrival frequency was either MEDIUM and HIGH. Since the tasks have loose deadlines, they can be processed whenever resources are available without detrimentally affecting the utility. However since the arrival frequency is medium to tightly constrained, there is a very high probability of overlapping tasks contending for resources. The arrival model information will allow the agent to dynamically adjust its decisions on tasks and use the bounded resources in an efficient way.

It can be deduced that the arrival model information available to the SHS is advantageous only in environments that are neither tightly constrained or loosely constrained. This is a characteristic shared with constraint satisfaction problems where there are few solutions in highly constrained problems and too many good solutions in loosely constrained problems. In either case, the difference between performance of alternative approaches is not significant.

Table 9 compares the percentage of tasks that were successfully completed by the four algorithms in different environments. In tightly constrained environments like those with tight task deadlines (AMT, AHT), the number of tasks completed is relatively low because often there aren't enough resources to execute the task and process all the external events also. In loosely constrained environments like ALM, CLL and ALL, task arrival is few and far between allowing the agent to complete one task successfully and to move on to the next task.

6.3 Multi-Agent Setup

An agent in a multi-agent setting makes decisions on the three events described in the single agent setup. Additionally, it reasons about two other events that occur specifically when coordination with another agent is required for completing the task. The following are the heuristics for the two additional meta-level decisions. The NHS is a myopic variant of these heuristics.

Environment#	SHS	NHS	Deter.	Rand.
AMM	41.08%	39.64%	30.52%	21.56%
AMT	28.60%	27.51%	21.7%	19.87%
AHT	22.08%	21.28%	12.47%	13.97%
ALM	56.86%	56.66%	55.09%	22.15%
AML	45.11%	39.61%	21.14%	21.27%
AHL	33.80%	28.75%	22.0%	19.04%
ALL	65.12%	63.39%	52.84%	23.48%
CLL	76.95%	71.78%	28.11%	24.51%

Table 9: Comparison of percent of tasks completed 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;

Table 10 describes SHS rules required to support each of the actions in Figure 5 for the *Presence of task requiring negotiation* event trigger. The event occurs when the *MetaNeg* information gathering action completes execution.

NTUG	NTDL	NLUG	NLDL	NLS	P(HPT)	MLC Decision
L	*	H	*	L	H/M	Drop Negotiation and Reschedule (B1)
H	*	L	*	H	L	Choose NegMech1 (B2)
M	*	L/M	*	H	L	Choose NegMech1 (B2)
H	M/LS	L	*	H	L/M	Choose NegMech2 (B3)

Table 10: SHS rules for *Presence of Task requiring Negotiation* event trigger. The column headers are NTUG = New Task Utility Goodness; NTDL = New Task Deadline;NLUG = Utility Goodness of Non-Local agent's task set; NLDL = Deadline of Non-Local task set; NLS = Slack in Non-Local schedule; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

Table 11 describes SHS rules required to support each of the actions in Figure 7 for the *Failure of negotiation* event trigger.

NTUG	NTDL	NLUG	NLDL	NLS	P(HPT)	MLC Decision
*	T	*	*	*	*	No ReNegotiation (C1)
M	*	L/M	*	H	L	Renegotiate using NegMech1 (C2)
H	LS/M	L	*	H	L/M	Renegotiate using NegMech2 (C3)

Table 11: SHS rules for *Failure of Negotiation* event trigger. The column headers are NTUG = New Task Utility Goodness; NTDL = New Task Deadline;NLUG = Utility Goodness of Non-Local agent's task set; NLDL = Deadline of Non-Local task set; NLS = Slack in Non-Local schedule; P(HPT) = Probability of Arrival of High Priority Tasks in the near future; MLC Decision = Meta-level Control Action Choice. The column values are the following : H = HIGH; M = MEDIUM; L =LOW; T = TIGHT; LS = LOOSE; * = all possible values; - = no value.

6.4 Multi-agent Experiments

This sub-section provides performance comparisons of the four different strategies to meta-level control: Naive Heuristic Strategy (NHS); Sophisticated Heuristic Strategy (SHS); Deterministic Strategy; and Random Strategy within a multi-agent context. The experimental setup is similar to the single agent set up except for the fact that two more decisions are added to the decision process, whether to negotiate with another agent about a non-local task and whether to renegotiate if a previous negotiation falls through.

Preliminary experimental results describing the behavior of two interacting agents is presented in Figure 11 and Table 12. Performance comparison of the various strategies in an environment, AMM, over a number of dimensions

Row#		SHS	NHS	Deter.	Rand.
1	AUG	111.44	89.84	77.56	45.56
2	σ	2.33	6.54	12.45	15.43
3	CT	9.21%	8.09%	14.28%	7.15%
4	RES	0%	14.28%	19.93%	1.49%
5	PTC	71.32%	56.34%	54.17%	57.78%
6	PTDEL	8.8%	3.98%	0%	59.96%

Table 12: Performance evaluation of four algorithms for two agents in a environment AMM 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); Row 5 is the percent of total tasks completed (PTC);Row 6 is percent of tasks delayed on arrival (PTDEL)

are provided. The results show that the combined utilities of the two agents when using the heuristic strategies is significantly higher than the combined utilities when using the deterministic and random strategies. The utility obtained from using SHS is significantly higher than NHS and also 14% more tasks are completed using SHS than the NHS. These preliminary results are encouraging since in this specific environment, the performance of the multi-agent system supports the hypothesis of this paper. Further experimental studies to establish the advantage of meta-level control in multi-agent systems are ongoing.

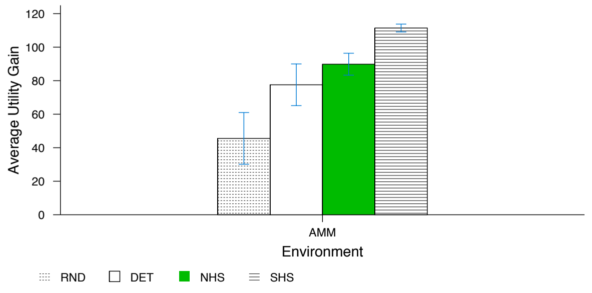


Figure 11: Average Utility Comparison between Heuristic Strategies and Baseline Strategies in a Multiagent environment. The error bars are one standard deviation above and below each mean

We have now presented two context sensitive heuristic strategies: the Naive Heuristic strategy (NHS) that uses myopic information to make meta-level control action choices; and the Sophisticated Heuristic strategy (SHS) that uses current state information and predictive information about the future to make non-myopic action choices. A description of the decision rules used in each of these strategies is provided. The experimental evaluation described in this section lead to the following conclusions : Meta-level control reasoning is advantageous in resource-bounded agents in different types of environments; the high-level features are good indicators of the agent state and facilitate effective meta-level control; the heuristic strategies establish the positive effects of meta-level control in resource-bounded agents because they outperform deterministic and random strategies; and predictive information about future arrival tasks is useful in some environments and not in others.

7 Reinforcement Learning Strategy

Can an agent automatically learn meta-level control policies for specific environments based on the high-level state information described in Section 3? Does this learned policy outperform the corresponding hand-crafted policy for that environment as described in Section 6? These are the two questions addressed in this section.

The high-level goal of this paper is to create agents which can maximize the social utility by successfully completing their goals. These agents also necessarily have limited computation, and detailed models of the task environments are not readily available. Reinforcement learning is useful for learning the utility of these control activities and decision strategies in such contexts. Section 7.2 describes the construction of a Markov decision process-based [31] meta-level controller which uses reinforcement learning techniques to approximate an optimal policy for allocating computational resources. This approach to meta-level control implicitly deals with opportunity cost as a result of the long-term effects of the meta-level decisions on utility. Sections 7.4 describes the complexities of the issues faced by multi-agent reinforcement learning agents. Experimental results describing the performance of the learned polices in both the single-agent and multi-agent cases are provided.

7.1 Reinforcement Learning

Reinforcement Learning [1, 19, 45, 46, 44, 51, 52] is a mathematical framework used by agents to learn how to map situations to actions so as to maximize a numerical reward signal. Supervised Learning (commonly used in research in machine learning, statistical pattern recognition and artificial neural networks) is learning from examples provided by a knowledgeable external supervisor. Reinforcement Learning is different from supervised learning in that the agent does not learn what actions to take from a "supervisor". Instead the usual approach taken by reinforcement learning agents involves discovering which actions yield the most reward by trying them out, associating expected reward values with different agent states, and using reward values to choose actions.

Two key features of reinforcement learning are the exploration-exploitation trade-off and credit assignment. A reinforcement learning agent, to maximize its reward, must prefer (exploit) actions which it has tried in the past and found to be effective in producing rewards. But to discover such actions, the agent has to try (explore) actions it has not selected before. The agent should be able to explore the action space to make better action selections in the future while at the same time progressively favor those actions that appear best.

The temporal credit assignment problem involves distributing rewards over a sequence of state-action pairs that lead up that reward. When actions are not rewarded immediately but receive a large positive or negative reward some time later, it is called delayed reinforcement. Reinforcement learning algorithms typically use a scheme for assigning the appropriate credit to all preceding state-action pairs after receiving a delayed reinforcement.

7.2 Single Agent Meta-Level Control

The learning approach adopted for the meta-level control problem is based on the algorithm developed in [42] where reinforcement learning is used in the design of a spoken dialogue system. Their problem is similar to the meta-level control problem in that it is also a sequential decision making problems and there is a bottle neck associated with collecting training data. As described in the experimental setup in Section 6, each episode lasting 500 simulation time clicks takes about 180 seconds on a Intel Pentium(R) machine with four 1.80 GHz processors running linux. It takes about 150 hours to obtain data from 3000 training episodes making data collection quite expensive.

As described in previous sections, the meta-level controller (MLC) in making its decisions does not directly use the information contained in the agent's current state. This would include quantitative information of tasks that are not yet scheduled, tasks that are partially executed, the schedule of the primitive actions that is to be executed as well as information of each primitive action for each time step. This leads to an exponential state space. Instead the MLC uses a set of high-level qualitative features which is constructed to abstract the real state information as much as possible without losing critical information. The advantage of this approach is that it simplifies the decision making process and provides the possibility for learning good rules.

The appropriate actions to take in each state are also defined. The reward function is determined by the utilities accrued by each completed domain task. The meta-level control policy is a mapping from each state to an action. 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 is implemented. Each episode is a sequence of alternating states, actions and rewards. As described in [42], the transition probabilities of the form $P(s'|s,a)$ are estimated, 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 MDP model representing system behavior for a particular environment is obtained from state set, action set, transition probabilities and reward function. Confidence in the accuracy 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 the optimal policy in the estimated MDP is determined using the Q-value version of the standard value iteration algorithm

[44]. 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 [44]:

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

When the value iteration algorithm converges[8], an optimal meta-level control policy (according to the estimated model) is obtained by selecting the action with the maximum Q-value at each state. The optimality of the policy depends on the accuracy with which the estimated MDP represents the particular environment. 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.
4. Compute an optimal meta-level control policy according to this MDP.
5. Reimplement the system using the learned meta-level control policy

7.3 Experiments

The experimental setup is as described in the previous sections. The training data for the RL strategy consisted of 3000 episodes. The training data as mentioned earlier is exploratory in that at each decision point one action from a set of allowable actions is chosen at random. After the 3000 episodes were completed, the estimated transition probabilities and reward function were determined. The meta-level control policy was determined using the Q-value version of value iteration as described above in the algorithm.The policy was then used on a test run consisting of 300 simulation test episodes.

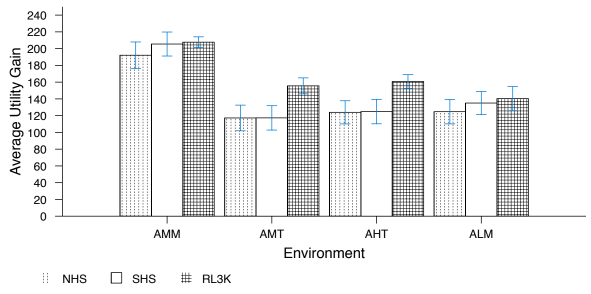


Figure 12: Utility Comparison of Learning Method to Heuristic Strategies for four different environments. The error bars are one standard deviation above and below each mean

The results described in Figure 12 show the utility accrued by the reinforcement learning, SHS and NHS strategies for four environments AMM, AHT, AMT and ALM. The data collection bottleneck described previously limited the number of environments considered to four. The four environments were chosen to represent problem classes where interesting behavior of meta-level control could occur: medium constrained environments (AMM, ALM) and tightly constrained environments (AHT, AMT). The following performance results are established experimentally:

1. In two of the environments, AMT and AHT, the RL strategy using the policy based on 3000 training episodes performed significantly better ($p < 0.05$) than the SHS with respect to utility and had significantly lower control duration.
2. In the two other environments, AMM and ALM, the RL strategy using the policy based on 3000 training episodes performed as well as (no significant difference at $p < 0.05$) than the SHS with respect to utility and had significantly lower control duration.

[8]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.

3. In loosely constrained environments like ALM, agents have enough resources to complete tasks successfully within the deadlines with out too much contention of resources.

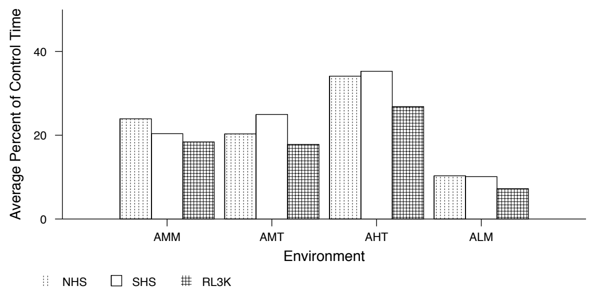


Figure 13: Control Time Comparison of Learning Method to Heuristic Strategies for four different environments

Figure 13 describes the percent of total time spent on control actions. The RL method spends significantly less time on control actions ($p < 0.05$) than the heuristic strategies in all four environments. The RL optimizes its actions in a non-myopic fashion since it can learn a more accurate model of the sequential decision making process than the heuristic strategies.

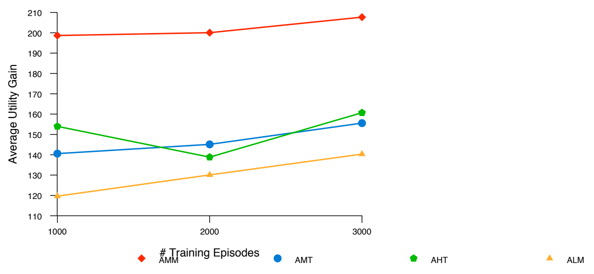


Figure 14: Relation of Average Utility to Increasing Training Data

Learning Curve Saturation: Figure 14 describes the effect of increasing training data on the performance of the learned policies. After every 1000 episodes, the cumulative transition probabilities and reward function were estimated and the corresponding policy was computed. This policy was then applied to 300 test episodes and the average results were computed. The performance of the agent improves with added training but the improvement does not increase proportionately with the training size. This seems to indicate that increased training data will not necessarily guarantee a monotonic improvement in performance and that the performance improvement will flatten out after a certain amount of training. This threshold is determined for each specific environment experimentally in this paper. 3000 seemed to be a good threshold for training size for the four environments described. The dip in the curve for AHT at episode 2000 is a local minima which occurred because of the tightly constrained environment. On deeper analysis of the data, we found that tasks in episode 2000 had extremely tight deadlines[9] and hence considerably fewer number of tasks could be completed. Figure 15 describes the relation of the percent of control durations to increasing training size.

Table 13 describes the actual values of the measures described in the preceding discussion.

Significance of discounting: γ in the dynamic programming formulation denotes the discount factor. The discount factor determines how much value is given to future rewards. When γ is set to 1.0, the agent gives a lot of importance to the long term effects of its current decision. When γ is set to 0.0, the agent does a one-step look ahead and is very myopic in its decision making. Figure 16 describes the utility gained by the agent after 3000 training episodes. Three meta-level control policies with γ set to 1.0, 0.5 and 0.0 are computed. These polices are then used to evaluate

[9]This is determined by the simulation environment.

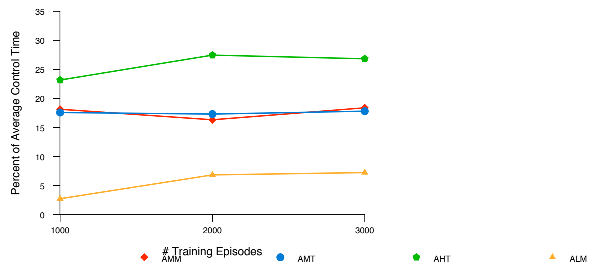


Figure 15: Relation of Control Time Durations to Increasing Training Data

Environment	RL-3000	RL-2000	RL-1000	SHS	NHS
AMM-UTIL	207.69	200.05	198.65	205.49	192.10
AMM-CT	18.39%	16.33%	18.14%	20.37%	23.92%
AMT-UTIL	155.56	145.11	140.57	117.34	117.25
AMT-CT	17.81%	17.31%	17.58%	24.95%	20.32%
AHT-UTIL	160.68	138.84	153.97	124.80	123.96
AHT-CT	26.83%	27.46%	23.17%	35.26%	34.09%
ALM-UTIL	140.32	130.09	119.64	135.05	124.74
ALM-CT	7.24%	6.84%	2.74%	10.11%	10.32%

Table 13: Utility and Control Time Comparisons over four environments; Column 1 is the environment type; Column 2, 3 and 4 represent the performance characteristics of the RL policy after 3000, 2000 and 1000 training episodes respectively; Column 4 and 5 represent the performance characteristics of SHS and NHS respectively;

300 test episodes and the average utilities over these 300 episodes are computed. Table 14 describes the values of the utility gained and the corresponding percent of control time for the three different polices. Column 1 is the type of environment, Column 2 describes the performance characteristics when the agent has a completely myopic view (γ=0.0), Column 3 describes the performance characteristics and control time when the agent has a partially myopic view (γ=0.5) and Column 4 describes the performance characteristics when the agent gives a lot of priority to long term effects of its decisions.

In medium constrained environments such as AMM and ALM, the average utility gained using the policy with γ set to 1.0 is significantly better ($p<0.05$) than the partially myopic policy with γ=0.5 and the myopic policy with γ=0.0. In tightly constrained environments such as AMT and AHT, the difference in performance of the non-myopic policy, the partially myopic policy and the myopic policy was not significant at the 0.05 level. These environments are so tightly constrained and are too dynamic to be able to effectively predict the future events and act on that information.

7.4 Multi-Agent Meta-Level Control

The agents in this domain are in a cooperative environment and have approximate models of the others agents in the multi-agent system. The agents are willing to reveal information to enable the multi-agent system to perform better as a whole. The interaction between 2 agents *R1* and *R2* is studied. The multi-agent aspect of the problem arises only when there is task requiring coordination with another agent. The agent rewards in this domain are neither totally positively correlated(team problem) nor are they totally negatively correlated(zero-sum game). Multi-agent reinforcement learning has been recognized to be much more challenging than single-agent learning, since the number of parameters to be learned increases dramatically with the number of agents. In addition, since agents carry out actions in parallel, the environment is usually non-stationary and often non-Markovian as well [26]. The experiments describe results on the convergence rates of the policies of the two agents in simple scenarios.

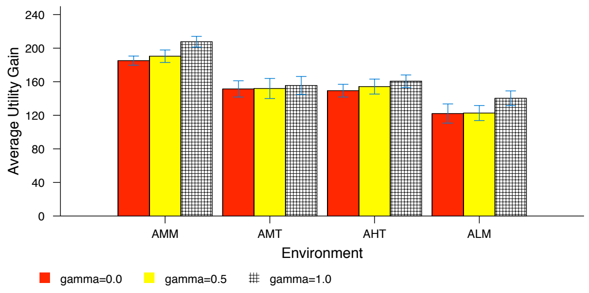


Figure 16: Utility Gains with varying discount rate ($\gamma = 0.0, 0.5, 1.0$). The error bars are one standard deviation above and below each mean

Environment	γ=0.0	γ=0.5	γ=1.0
AMM-UTIL	185.19	190.46	207.69
AMM-CT	19.56%	18.63%	18.38%
AMT-UTIL	151.48	151.99	155.56
AMT-CT	17.95%	17.90%	17.80%
AHT-UTIL	149.40	154.26	155.56
AHT-CT	26.42%	26.23%	17.81%
ALM-UTIL	122.16	122.74	140.32
ALM-CT	6.88%	6.90%	7.24%

Table 14: Comparison of utility gain and percent of control time for four different environments while varying the discount rate ($\gamma = 0.0, 0.5, 1.0$)

7.5 Experiments

The meta-level control decisions that are considered in the multi-agent set up are: when to accept, delay or reject a new task, how much effort to put into scheduling when reasoning about a new task, whether to reschedule when actual execution performance deviates from expected performance, whether to negotiate with another agent about a non-local task and whether to renegotiate if a previous negotiation falls through. For all the experiments the following costs are assumed. The meta-level control actions have an associated cost of 1 time unit; the drop task and delay task actions take 1 time unit also. The decision to negotiate and whether to renegotiate also take 1 unit of time. The call to simple scheduler costs 2 time units and the cost of computation of complex features costs 2 time units, the cost of detailed scheduling tasks with less than five methods is 4 units, with less than ten methods is 12 time units and greater than ten methods is 18 time units[10].

The task environment generator in the multi-agent setup also randomly creates task structures while varying three critical factors:

1. complexity of tasks $c \in \{simple(S), complex(C), combination(A)\}$
2. frequency of arrival $f \in \{high(H), medium(M), low(L)\}$
3. tightness of deadline $dl \in \{tight(T), medium(M), loose(L)\}$.

Complexity of tasks as described earlier refers to the expected utilities of tasks and the number of alternative plans available to complete the task. A simple task, in the multi-agent setup, has two primitive actions and its structure and number of possible alternatives is similar to the AnalyzeRock task (Figure 1) described in Section 2. The utility distribution and duration distribution of a simple task is within a 5% range of the corresponding distributions of AnalyzeRock. A complex task in the multi-agent set-up can be of two types, one has structure similar to ExploreTerrain

[10]The non-linear increase in processing time is due to the design of the heuristic complex scheduler. The Design-to-Criteria scheduler is shown to work well for tasks with fewer than 10 primitive actions[50].

Environment	RL-3000	SHS	NHS
AMM-UTIL	118.56	111.44	89.84
AMM-CT	8.86%	9.21%	8.09%

Table 15: Utility and Control Time Comparison for a multi-agent environment; Column 1 is the environment type; Column 2 represents the performance characteristics of the RL policy after 3000 training episodes; Column 3 and 4 represent the performance characteristics of SHS and NHS respectively;

task described in Figure 1 and the other has structure similar to that of GetImage task described in Figure 8. The utility distribution and duration distribution of a complex task is within a 5% range of the corresponding distributions of ExploreTerrain or GetImage task. The combination value means that 50% of the tasks are simple and 50% are complex tasks. The frequency and deadline tightness are the same as in the single agent setup.

Preliminary experimental results describing the behavior of two interacting agents is presented in Figure 17 and Table 15. Agent *R1's* was fixed to the best policy it was able to learn in the single agent environment. Agent *R2* then learned its meta-level control policy within these conditions.

Performance comparison of the heuristic strategies to the RL strategy in a single environment, AMM, is provided. The results show that the combined utilities of the two agents when using the RL strategy is as good as the SHS strategy which uses environment characteristic information in its decision making process. The RL strategy also learns policies which significantly outperform the NHS strategy in this environment. The performance of the multi-agent system supports the hypothesis of this paper.

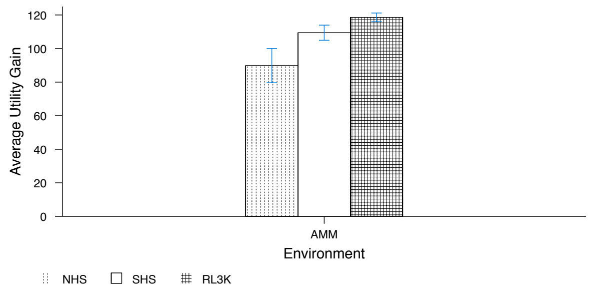


Figure 17: Average Utility Comparison between Heuristic Strategies and RL Strategy (3000 training episodes) in a Multiagent environment. The error bars are one standard deviation above and below each mean

We have described a reinforcement learning approach which equips agents to automatically learn meta-level control policies. The empirical reinforcement learning algorithm used is a modified version of the algorithm developed by [42] for a spoken dialog system. Both problem domains have the bottle neck of collecting training data. The algorithm optimizes the meta-level control policy based on limited training data. The utility of this approach is demonstrated experimentally by showing that the meta-level control policies that are automatically learned by the agent perform as well as the carefully hand-generated heuristic policies. The sequential effects of the problem domain was verified by showing that varying the value of future rewards significantly affects the agent's performance.

8 Related Work

There has been enormous amount of work on intelligent agent control e.g. [3, 10, 27, 50, 55]. These systems describe flexible and goal-directed mechanisms capable of recognizing and adapting to environmental dynamics and resource bounds. The emphasis in these works is to build an adaptive control layer which reasons about domain-level costs. They do not, however, explicitly reason about the control costs. The meta-level control architecture described in this paper reasons explicitly about control costs and includes reasoning about costs at all levels of computation.

8.1 Procedural Reasoning System

The Procedural Reasoning System (PRS) [11] is a hybrid system, where beliefs are expressed in first-order logic and desires represent system behaviors instead of fixed goals. It is an architecture for embedded systems that need to deliberate in real-time. A PRS agent consists of a database of the system's current beliefs, a set of current goals, a library of plans (called knowledge areas or KAs) and an intention structure. The KAs describe sequences of actions and tests that can be performed to meet a goal or react to a situation. The intention structure consists of a partially ordered set of those plans chosen for execution. An interpreter works with these components to select an appropriate KA based on beliefs and goals, place that plan in the intention structure and execute it. Meta-level KAs are functionally similar to the meta-level control layer in the agent architecture presented in this paper. The meta-level KAs are used to decide among multiple applicable domain KAs in a particular situation, reason about failure to satisfy goals, and manage the flow of control among intentions (including determining when to continue applying meta-level KAs versus executing the current domain-level plan). KAs are interruptible when external events cause changes to the database, thus allowing rapid response to changing environmental situations. PRS can be configured to respond to world events within a bounded amount of time. PRS is not concerned with cost of meta-level reasoning explicitly. It would probably be advantageous for the PRS architecture to reason about costs at all levels and not only at level of the domain KAs.

8.2 Guardian

Hayes-Roth [15] describes an opportunistic control model that can support different control modes expected of an intelligent agent. The control model handles multiple goals, limited resources, and dynamic environments. She argues that in dynamic environments, it is often necessary to make decisions that may not be optimal, but are rather satisfactory under the current conditions. The meta-level control work described in this paper similarly computes approximate solutions rather than optimal solutions.

The system she develops that solves problems closest to the complexity to the problems we are interested in is Guardian [16]. It is an experimental intelligent agent based on a blackboard architecture for monitoring patients in a surgical ICU. The agent consists of a manager that filters and processes inputs, a satisficing control cycle to bound the amount of time spent doing meta-level reasoning, and an anytime diagnosis component. Large amounts of input data arrive at the agent periodically. Much of this is low level data that just confirms current patterns, but occasionally important or unexpected information arrives. The input manager dynamically builds and modifies filters to sending new important information to be processed by the reasoning component while not overburdening it with needless detail as problem solving progresses. High-level control takes the form of plans that are dynamically created at runtime by control knowledge sources. For an actual application constructing these control knowledge sources and control plans can be a major tasks. They emphasize that such dynamic construction is necessary because of the changing requirements of the filters in different problem solving situations.

Guardian has an agenda based control mechanism. The control cycle chooses the best action to perform by processing actions most likely to be rated highly first. As soon as an action is found that is good enough or the time limit for control reasoning has run out, the best action found so far is recommended. The anytime diagnosis component diagnoses and recommends treatment for medical conditions in the patients being monitored. This component uses action based hierarchies, which are similar to decision trees, and each node in the hierarchy is a collection of faults, human error or machine failure, with associated action to take.

These components give Guardian the ability to be reactive in situations that require it, by using input filters to separate out important data, and a satisficing control cycle to quickly determine how to respond to it. Guardian, however, is not equipped with an overall planning mechanism to guide its real-time behavior. It does not reason about long-term effects of choices explicitly.

8.3 Bounded Rationality and Meta-level Control

Flexible, autonomous systems in complex environments generally require the ability to reason about resource allocation to computation at any point in time. Doyle's 'rational psychology' project [9] is based on the idea that computations, or state changes, are also actions to be reasoned about. He used the idea of bounded rationality in the context of beliefs, intentions and learning. Horvitz [18] also studied rational choice of computation in the context of designing intelligent systems.

The basic idea of bounded rationality arises in the work of Simon with his definition of procedural rationality [39]. Simon's work has addressed the implications of bounded rationality in the areas of psychology, economics and artificial intelligence [41]. He argues that people find satisfactory solutions to problems rather than optimal solutions because people do not have unlimited processing power. In the area of agent design, he has considered how the nature of the environment can determine how simple an agent's control algorithm can be and still produce rational behavior.

In the area of problem solving, Simon and Kadane [40] propose that search algorithms for finding solutions to problems given in terms of goals are making a trade-off between computation and solution quality. A solution that satisfies the goals of a problem is a minimally acceptable solution.

Good's type II rationality [12] is closely related to Simon's ideas on bounded rationality. Type II rationality, which is rationality that takes into account resource limits, is a concept that has its roots in mathematics and philosophy rather than psychology. Good creates a set of normative principles for rational behavior that take computational limits into account. He also considers explicit meta-level control and how to make decisions given perfect information about the duration and value of each possible computation.

In order to make the trade-offs necessary for effective meta-level control, the meta-level controller needs some method for predicting the effect of more computation on the quality of a plan. One method for doing this is to use a performance profile. The idea comes from the study of anytime algorithms. Anytime algorithms can be interrupted at any point to return a plan that improves with more computation [6]. The performance profile gives the expected improvement in a plan as a function of computation time.

An alternative to using performance profiles is to use the performance of the planner on the current problem to predict the future. Nakakuki and Sadeh use the initial performance of a simulated annealing algorithm on a machine shop scheduling problem to predict the outcome for a particular run [29]. They have found that poor initial performance on a particular run of the algorithm is correlated with poor final performance. This observation is used to terminate unpromising runs early and restart the algorithm at another random initial state.

Anytime algorithms can be combined to solve complex problems. Zilberstein and Russell [54] look at methods for combining anytime algorithms and performing meta-level control based on multiple performance profiles. Combining anytime algorithms produces new planning algorithms that are also characterized by a performance profile. Compilation techniques described in [55], can be used to compile programs consisting of both anytime and traditional algorithms [11]

Hansen and Zilberstein [13] extend previous work on meta-level control of anytime algorithms by using a non-myopic stopping rule. It finds an intermediate strategy between continuous monitoring and not monitoring at all. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. This work has significant overlap with the foundations of the meta-level control reasoning framework described in this paper. It deals with the single meta-level question of monitoring and considers the sequential effects of choosing to monitor at each point in time. It keeps the meta-level control cost low by using a lookup-table for the policy.

Harada and Russell [14] describe initial work where the computational process is explicitly modeled. It provides initial ideas for using search as the model of computation in the Tetris domain. They propose the use of Markov Decision Processes and reinforcement learning as their solution approach. This work was not pursued further[12]. The methodology in this research was developed independently of their effort. It was built for a complex domain where the meta-level decisions have down-stream effects. The domain is characterized by uncertainty in action durations and utility accrued

8.4 Reinforcement Learning in Agent Systems

Algorithms for sequential Reinforcement Learning (RL) tasks have been studied mainly within a single agent context [1, 46, 51]. Some of the later work described below have applied RL methods such as Qlearning to multi-agent settings. In many of these studies, the agents learn about either simple dependent tasks or independent tasks. Sen et al. [38] describe 2-agent block pushing experiments, where the agents try to make the block follow a line by independently applying forces to it. Tan [48] reports on grid-world predator-prey experiments with multi-agent RL, focusing on the sharing of sensory information, policies, and experience among the agents. Unfortunately, just slightly harder predator-prey problems [37] and prisoner's dilemma [36] have uncovered discouraging results. The standard Q-learning algorithms are not guaranteed to converge in non-stationary environments where all agents are learning simultaneously. The agents had to keep detailed accounts of their entire history and interaction patterns, in addition to implementing long exploration schedules to achieve convergence.

[11]The performance profile of a traditional algorithm is presumably a single step function.

[12]Personal communication with second author.

Crites and Barto [5] apply multi-agent RL algorithms to elevator dispatching, where each elevator car is controlled by a separate agent. The agents don't communicate with each other and an agent treats the other agents as a part of the environment. The problem is complicated by the fact that their states that are not fully-observable and they are non-stationary due to changing passenger arrival rates. Littman and Boyan [24] describe a distributed RL algorithm for packet routing, using a single, centralized Q-function, where each state entry in the Q-function is assigned to a node in the network which is responsible for storing and updating the value of that entry. In our work, the entire Q-function, not just a single entry, is stored by each agent. Littman [25] experiments with Q-learning agents that try to learn a mixed strategy that is optimal against the worst possible opponent in a zero-sum 2-player game.

Lagoudakis and Littman [22] describe a RL-based approach 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 the meta-level control problem although they only reason about a single problem instance at any point in time. The sequential nature of the decision process in our work complicates the reasoning process. Other multi-agent learning research has used purely heuristic algorithms for complex real-world problems such as learning coordination strategies [43] and communication strategies [20] with varying success.

The meta-level control architecture described in this paper differs from the above mentioned works in that it uses RL to make meta-level control decisions in a complex sequential decision making, cooperative multi-agent environment. It emphasizes the necessity for alternative ways of performing computations and it dynamically reasons about the cost of computation based on the current context.

Many researchers in AI have addressed the need for abstractions to solve large-scale planning problems. Abstraction is the process by which a system simplifies its decision making process by choosing only the information relevant to decision making process and ignoring the irrelevant information. In the RL literature, temporal abstraction and hierarchical control have been used to combat the curse of dimensionality in a principled way. The aim of hierarchical RL is to discover and exploit hierarchical structure within a Markov decision problem. The options formalism of Sutton, Precup and Singh [47] describes closed-loop policies for taking action over a period of time. They show that options can be used interchangeably with primitive actions in both planning methods and learning methods. The foundation of the theory of options is provided by the existing theory of Semi-Markov Decision Processes (SMDPs) and associated learning methods. Parr and Russell [30] developed an approach to RL in which the policies considered are constrained by hierarchies of partially specified machines. This allows for the use of prior knowledge to reduce the search space. The SMDP -based framework allows knowledge to be transferred across problems and for component solutions to be recombined to solve larger and more complicated problems. The MAXQ framework of Dietterich [8] relies on creating a hierarchy of SMDPs whose solutions can be learned simultaneously. He shows that hierarchical RL using the MAXQ framework can be much faster and more compact than flat RL. He also shows that recursively optimal policies can be decomposed into recursively optimal policies of individual subtasks and these subtask policies can be re-used wherever the same subtask arises.

These works emphasize the importance and advantages of abstraction in RL. The meta-level control work however is different from these works because it uses abstract representation of the state based on the similarity of states. In other words, A number of the agent's real states are represented by a single abstract state because of their similarity of their feature values (excluding time) which is different from the temporal abstractions described in the above three works.

9 Conclusions

This paper explores the issue of meta-level control in complex agents situated in social and dynamic environments. As discussed in the introduction, complex agents can concurrently perform several different goals of varying worth and deadlines, dynamically choose alternate ways to achieve these goals and make choices on how much effort to spend on deliberative actions. Deliberations about the tasks may involve resource-intensive computation. Also, the control decisions made by the agent may have down-stream effects on the availability of resources and processing available to future tasks. Meta-level control is the ability of an agent to optimize its long-term performance by choosing and sequencing its deliberation and execution actions appropriately. It reasons about the cost of computation at all levels as a first-class entity.

This paper establishes the following hypothesis: *Meta-level control with bounded computational overhead allows complex agents to solve problems more efficiently in dynamic open multi-agent environments. Meta-level control is computationally feasible through the use of an abstract representation of the agent state. This abstraction concisely cap-*

tures critical information necessary for decision making while bounding the cost of meta-level control and is appropriate for use in automatically learning the meta-level control policies.

Main Results

A meta-level agent architecture for bounded-rational agents which supports alternative approaches for deliberative computation is described. The meta-level control has limited and bounded computational overhead and supports reasoning about about costs of planning, scheduling and negotiation as first-class entities. Accounting for costs of reasoning at all levels is necessary for guaranteeing the performance characteristics of real-time systems. An experimental testbed to evaluate the agent performance was set up using the MASS simulation environment where the architecture described was fully implemented. Tasks of varying complexity were used to study the performance of the architecture using various policies for meta-level control. A deterministic policy was used as a base-line for evaluation. An agent while deciding to trade-off deliberation versus execution action is in effect reasoning on whether to retain control of its resources or to decide to perform a task to gain the associated utility while at the same time giving up control of the required resources. One of the interesting contributions of this work is the way it exploits knowledge of the tasks from the task structures. The state features are computed using thresholds which are specific to the task being analyzed.

This paper establishes that meta-level control in resource-bounded rational agents is beneficial using empirical evidence. Two context sensitive hand-generated heuristic strategies are defined: the Naive Heuristic strategy (NHS) that uses myopic information to make meta-level control action choices; and the Sophisticated Heuristic strategy (SHS) that uses current state information and predictive information about the future to make non-myopic action choices. The heuristic strategies significantly outperform ($p<0.05$) deterministic and random strategies by about 30% on average confirming the importance of meta-level control. We also experimentally show that a few abstract features which accurately capture the state information and task arrival model enable the meta-level control component to make computationally-bound decisions which significantly improve agent performance.

An observation made from the experiments is that the cost of control actions in terms of resources used is an important factor in determining the need for meta-level control. Meta-level control is advantageous in environments where the control costs are high enough so that the resources available for domain actions are significantly constrained. When the cost of control actions becomes significantly inexpensive in a non-stationary environment, the hand-generated rules have to be rewritten to account for this fact. The learning method, on the other hand, can automatically construct a policy offline which adapts to the new costs.

This work also provides insight into the usefulness of reinforcement algorithms in complex multi-agent sequential decision-making problems. A reinforcement learning approach which equips agents to automatically learn meta-level control policies is described. This empirical algorithm is a modified version of the algorithm developed by [42] for a spoken dialog system. Both problem domains have the bottle neck of collecting training data. The algorithm optimizes the meta-level control policy based on limited training data consisting of 3000 runs. The utility of this approach is demonstrated experimentally by showing that the meta-level control policies that are automatically learned by the agent perform as well as if not better than the carefully hand-generated heuristic policies at the $p<0.05$ level. One surprising and useful result was that the agents were able to learn useful meta-level control policies with a small amount of training (3000 episodes). The sequential effects of the problem domain were verified by showing that varying the value of future rewards significantly affects the agent's performance.

Applying this work

This paper shows that meta-level control can be effective in real-time environments, characterized by uncertainty and limited computational resources. In these environments, computational commodities such as time, memory, or information can be traded for gains in the value of computed results. It also shows that efficient and inexpensive meta-level control which reasons about the costs and benefits of alternative computations leads to improved agent performance in resource-bounded environments. This is a flexible, run-time approach which seeks to optimize rather than satisfice solution quality.

This work also shows that a meta-level control policy can be learned in a non-deterministic, inaccessible and model-free environment. In an inaccessible environment, an agent must maintain some internal state to try to keep track of the environment, since it is not possible for states to identified just based on percepts. The learning strategy allows for meta-level control in uncertain environments whose model is not available. The empirical reinforcement learning algorithm allows the agent to construct a partial model of the environment and use the information to define effective action policies.

Additionally, the paper has identified scenarios in which predictive information about future task arrivals has limited utility. If the environment is characterized by high frequency of arrival of tasks with tight deadlines, then the meta-level controller will constantly have to reevaluate its decisions every time a new task arrives. These decisions are valid when made within a myopic context because of the dynamic environment. Hence predictive information about the future does not necessarily improve performance. If the environment is characterized by low frequency of arrival of tasks and the tasks have loose deadlines, then the environment is loosely constrained. In such environments, the downstream effects of decisions is minimal, since the tasks are spaced out enough so that there is minimal contention of resources by multiple tasks. This means predictive information about the future does not provide any additional performance advantage.

We plan to extend this work by introducing more complex features which will make the reasoning process more robust. 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.

10 Acknowledgments

We would like to thank Shlomo Zilberstein for his help in constructing the model described in Section 2 and Andy Barto for his valuable comments on the Reinforcement Learning Algorithm. We also thank Claudia Goldman and several anonymous reviewers for their useful comments.

"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 P00005. 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 the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory or the U.S. Government."

APPENDIX

Detailed Execution Trace of R1's Decision Making Process

In order to clarify the details of the approach, a detailed time-line execution trace of a sample run of the example in Section 2 is provided. It provides a detailed view of the meta-level reasoning process of agent *R1* based on abstract state features. It describes how the different components of the architecture interact with each other and details the various meta-level and control-level decisions taken for a particular set of environmental conditions. Agent *R1's* tasks are described in Figure 1. Suppose that along with information of the task structures, agent *R1* also receives abstractions of tasks *AnalyzeRock* and *ExploreTerrain*. Abstractions for the tasks are described in the following table.

Alternative	Method Sequence	EU	ED	NCT
$AnalyzeRock^1$	{GettoRockLocation}	6	8	0
$AnalyzeRock^2$	{ FocusSpectrometeronRock}	10.2	10.2	0
$AnalyzeRock^3$	$\{GettoRockLocation, FocusSpectrometeronRock\}$	16.2	18.2	0
$ExploreTerrain^0$	{ExamineTerrain}	12	8	0
$ExploreTerrain^1$	$\{MetaNeg, NegMech1, CollectSamples\}$	12.6	16.2	10.2
$ExploreTerrain^2$	$\{MetaNeg, NegMech2, CollectSamples\}$	13.05	16.6	10.2
$ExploreTerrain^3$	$\{MetaNeg, NegMech1, ExamineTerrain, CollectSamples\}$	24.6	24.2	10.2
$ExploreTerrain^4$	$\{MetaNeg, NegMech2, ExamineTerrain, CollectSamples\}$	25.05	24.6	10.2

Figure 18: Abstraction information for tasks AnalyzeRock and ExploreTerrain. The columns respeectively represent the alternative name, method sequence of alternative, expected utility of alternative (EU), expected duration of alternative (ED) and Non-Computational Time(NCT, time for non-local methods)

R1 will address several of the meta-level decisions listed below:

1. Should the method *CollectSamples* of task *ExploreTerrain* which is enabled by *R2* be included in *R1*'s schedule? This will determine the choice of the abstract alternative.
2. Should *R1* reason about incoming tasks at their arrival times or later?
3. What extent of reasoning should be invested in each task? Should it involve a reschedule action?
4. When a decision is made to schedule a task or set of tasks, the following parameters need to be determined. Where is it most appropriate to include slack in the schedule/policy to allow meta-level reasoning of unanticipated events? How much effort must be put into scheduling the task(s) by the scheduler?

Task structure *ExploreTerrain* contains a non-local enables and is translated to contain virtual nodes which represent meta-level activity as shown in Figure 19. Method *CollectSamples* has an incoming enables the following transformation rule is applied to it.

Transformation rule: If task/method X (*CollectSamples* in example) has an incoming external enables, replace it by task X'(*CollectSamples'*). Task X' can be achieved by first executing the *MetaNeg* primitive action which is a local quick and inexpensive analysis done by the negotiation initiating agent to determine whether or not its worthwhile to include the task/method with the negotiation overhead in the schedule. It involves some low-level information gathering and the decision on negotiation is made based on the agent's own load, the task utility, the profiles of the other tasks the local agent has to perform and the load of the enabling agent. The *MetaNeg* method enables the *NegTask* task which stands for Negotiation Task. It has very low utility by itself but is an important activity since it is a pre-condition for other methods with non-zero utilities. The information gathered by the *MetaNeg* action is used to decide whether to continue negotiation. If a decision to continue negotiation is made, the information gathered is also used to choose one of the two negotiation mechanisms *NegMech1* or *NegMech2*. *NegMech1* is a single shot negotiation mechanism and represents the quick alternative which has a lower probability of succeeding. *NegMech2* is the multi-step alternative which takes longer duration and has a higher probability of success as proposals and counter-proposals are handled. Successful execution of *NegTask* leads to the enablement of method X(*CollectSamples* in this example) as shown in the figure.

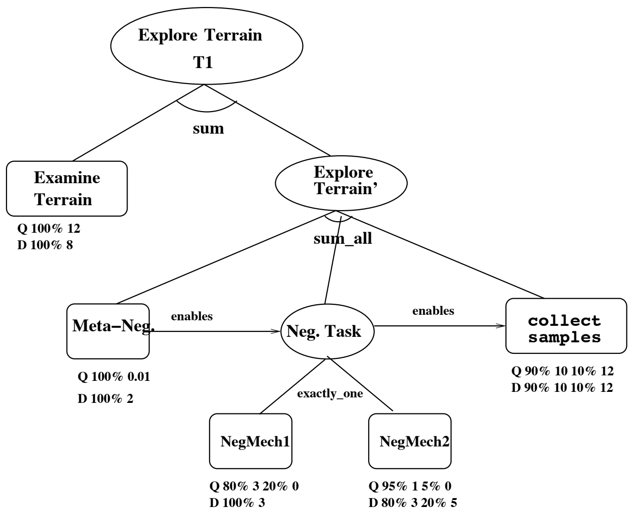


Figure 19: Task ExploreTerrain modified to include meta-negotiation action

Consider a scenario where the arrival model for agent *R1* (described as $TaskName < ArrivalTime, Deadline >$) is as follows:

1. $AnalyzeRock < AT = 1, DL = 40 >$,
2. $ExploreTerrain < AT = 15, DL = 80 >$,
3. $AnalyzeRock < AT = 34, DL = 90 >$,

1	2	3	4	5	6	7	8	9	10	11	12
ID	CT	NewTaskList	Agenda	ScheduleList	ExecutionList	IG	U_a^s	D_a^s	U_a^i	D_a^i	U^t
S0	1	ϕ	ϕ	ϕ	ϕ	ϕ	0	0	0	0	0
S1	2	$T0 < 1, 40 >$	ϕ	ϕ	ϕ	ϕ	0	0	0	0	0
S2	3	ϕ	ϕ	$T0 < 1, 40 >$	ϕ	ϕ	0	0	0	0	0
S3	7	ϕ	ϕ	ϕ	$\{M1, M2\}$	ϕ	0	0	0	0	0
S4	13	ϕ	ϕ	ϕ	$\{M2\}$	ϕ	6	8	0	0	6
S5	16	$T1 < 15, 80 >$	ϕ	ϕ	$\{M2^{exe}\}$	ϕ	6	8	0	2	6
S6	17	ϕ	$T1 < 15, 80 >$	ϕ	$\{M2^{exe}\}$	ϕ	6	8	0	2	6
S7	25	ϕ	$T1 < 15, 80 >$	ϕ	ϕ	ϕ	18	18	0	0	18
S8	26	ϕ	$T1 < 15, 80 >$	ϕ	ϕ	ϕ	0	0	0	0	18
S9	27	ϕ	ϕ	$T1 < 15, 80 >$	ϕ	ϕ	0	0	0	0	18
S10	31	ϕ	ϕ	ϕ	$\{MetaNeg, NegMech2, M3, M4\}$	ϕ	0	0	0	0	18
S11	33	ϕ	ϕ	ϕ	$\{NegMech2, M3^{exe}, M4\}$	<H,L,H>	0	1	0	2	18
S12	35	$T0 < 34, 90 >$, $T1 < 34, 90 >$	ϕ	ϕ	$\{NegMech2, M3^{exe}, M4\}$	ϕ	0	3	0	2	18
S13	36	ϕ	ϕ	$T0 < 34, 90 >$, $T1 < 34, 90 >$, $T1^{exe} < 15, 80 >$	ϕ	ϕ	0	2	0	2	18
S14	43	ϕ	ϕ	ϕ	$\{NegMech2, M3^{exe}, M4, M1, M2\}$	ϕ	0	3	0	2	18
S15	46	ϕ	ϕ	ϕ	$\{M3^{exe}, M4, M1, M2\}$	ϕ	1	7	0	3	19
S16	52	ϕ	ϕ	ϕ	$\{M4, M1, M2\}$	ϕ	9	15	0	0	27
S17	58	ϕ	ϕ	ϕ	$\{M4, M2\}$	ϕ	15	21	0	0	33
S18	65	ϕ	ϕ	ϕ	$\{M4, M2^{exe}\}$	ϕ	21	27	6	6	39
S19	77	ϕ	ϕ	ϕ	$\{M2^{exe}\}$	ϕ	33	39	6	6	51
S20	80	ϕ	ϕ	ϕ	ϕ	ϕ	0	0	0	0	57

Figure 20: Agent *R1's* state transitions

4. $ExploreTerrain < AT = 34, DL = 90 >$.

So task *AnalyzeRock* arrives at agent *R1* at time 1 with a deadline of 40, task *ExploreTerrain* arrives at time 15 with a deadline of 80 and so on. The goal is to maximize the expected utility over this deadline. The finite horizon considered for this scenario is 100 time units (D=100).

Figure 20 describes the *real* states agent *R1* is in while executing the best policy prescribed by the meta-level controller. The agent visits 20 states of the four million possible states while following the prescribed policy. The rows represent the system states in sequential order. Column 1 is the state identity. Columns 2-11 represent the dynamic state variables. Column 2 represents Current Time, Columns 3, 4, 5 and 6 represent the NewTaskList, the Agenda, the ScheduleList, and the ExecutionList respectively. Column 7 represents the information gathered upon execution of the *MetaNeg* information. It is in the form of a 3-tuple describing Expected Utility, Expected Deadline Tightness, Slack Amount information of the other agent's (*R2*) schedule. Column 8 represents the utility accrued by current schedule at current time. Column 9 is the duration spent on current schedule at current time. Column 10 and 11 respectively represent the utility accrued and duration spent by the executing primitive action when it is interrupted and control switches from execution to meta-level control component. Column 12 represents the total utility accrued by the agent at current time.

To keep the representation concise, Task *AnalyzeRock* is called task *T0*, Task *ExploreTerrain* is called task *T1*,

The corresponding states of *R1* are provided in the time-line description of the execution history following the figure. Only the features whose values are not "NONE' and whose values have changed since the previous state are mentioned in the state description.

Time 1: *R1* is in state **SO** since it has no current tasks and is in a waiting state. Task $AnalyzeRock < 1, 40 >$ arrives with a deadline of 40. Meta-level controller is invoked to determine the new state. Suppose the agent is given the information that there is a MEDIUM probability of high priority task arriving in the near future (Feature F5).

Time 2: *R1* is in state **S1** has the following features:
F0:$< 1, 0, 0, 0 >$
A single new task has arrived.
F1: HIGH
The new task has highest utility gain since agenda, scheduling list and execution list are empty.
F2: TIGHT
New task has closest deadline since agenda, scheduling list and execution list are empty.
F5: MEDIUM
There is a medium probability of a valuable task arriving. This is prior knowledge available to the agent. Based on the state information, the meta-level control policy prescribes the action **Call Detailed Scheduler**. The meta-level controller computes the features for the resulting state.

Time 3: *R1* is in state **S2** and has the following features which determine the parameters for scheduling:
F0:$< 0, 0, 1, 0 >$
F1: HIGH
F2: TIGHT
It is not necessary to analyze local agent's available slack since scheduling and execution lists are empty. Based on the new state information and arrival model, the meta-level control policy prescribes action to **Begin scheduling with the following parameters to the scheduler TSF=2; E=2, S=10%**.

Time 7: *R1* is in state **S3** when the scheduler emits the following schedule **{GettoRockLocation, FocusSpectrometeronRock}**. The best action for this state is to **Begin Execution of GettoRockLocation**.

Time 13: *R1* is in state **S4** when execution of the method **GettoRockLocation** completes with utility/reward of 6. The features are
F8: LOW
The deviation from expected performance is very small.
The best action for this state is to **Begin Execution of FocusSpectrometeronRock.**

Time 15: Execution of method **FocusSpectrometeronRock** is interrupted and control switches from the execution component to the meta-level control component when task $ExploreTerrain < 15, 80 >$ arrives with a deadline of 80.

Time 16: *R1* is in state **S5** which has the following features:
F0: $< 1, 0, 0, 1 >$
A single new task has arrived and execution list is non-empty exists,. Agenda and Scheduling list are empty.
F1: HIGH
Based on the task abstraction, it is deduced that the utility of the new task is always higher than the lowest possible utility of the current task set.
F2: LOOSE
Based on the task abstraction, the deadlines of new task is far enough in the future to schedule any of the alternatives of that task **ExploreTerrain** after the current task is completed.
Based on the state information, the meta-level control policy prescribes the action **Add to agenda**

Time 17: *R1* is in state **S6** which has the following features:
F0: $< 0, 1, 0, 1 >$
The best action prescribed is **Resume execution of interrupted method.**

Time 25: *R1* is in state **S7** when method **FocusSpectrometeronRock** completes with utility/reward of 12 and task **AnalyzeRock** completes with total utility/reward of 18. The features are
F8: LOW
The deviation from expected performance is very low .
Since the scheduling and execution lists are empty, the agenda is automatically checked and task $ExploreTerrain < 15, 80 >$ is retrieved. The best action is **Evaluate task in agenda**.

Time 26: *R1* is in state **S8** which has the following features:
F0: $< 0, 1, 0, 0 >$
Agenda has a single item in it.

F1: HIGH
Task on the agenda has highest utility gain since it is the only task to be reasoned about by the agent.
F2: MEDIUM
The deadline is not too close nor too far off.
F3: NONE
F4: NONE
The current schedule is reset to empty. Based on the state information, the meta-level control policy prescribes the action **Call Detailed Scheduler**.

Time 27: *R1* is in state **S9** which has the following features:
F0: $< 0, 0, 1, 0 >$
Based on the new state information, the meta-level control policy prescribes action to **Begin scheduling with the following parameters to the scheduler TSF=2, E=2, S=30%.**

Time 31: *R1* is in state **S10** when the scheduler emits the following schedule **{MetaNeg, NegMech2, ExamineTerrain, CollectSamples}**. The best action for this state is to **Begin Execution of MetaNeg in parallel with execution of ExamineTerrain**.

Time 33: *R1* is in state **S11** when execution of **MetaNeg** completes. *R1*'s total utility is still 18. The information gathered by **MetaNeg** is as follows: *R2* is executing schedule with HIGH expected utility/reward and the deadline tightness for these tasks is LOW. There is also high slack in *R2's* schedule. The following features are set:
F7: HIGH
R2 has high amount of slack.
F12: HIGH
The non-local agent can easily fit the new task's enabler in its schedule.
Based on the state information, the meta-level control policy prescribes the action **Choose NegMech2 and continue.** Method **NegMech2** and **FocusSpectrometeronRock** are initiated in parallel.

Time 34: Execution of method **FocusSpectrometeronRock** is interrupted and control switches from the execution component to the meta-level control component when tasks $AnalyzeRock < AT = 34, DL = 90 >$ $, ExploreTerrain < AT = 34, DL = 90 >$ arrive.

Time 35: *R1* is in state **S12** which has the following features.
F0: $< 2, 0, 0, 1 >$
F1: HIGH
F2: TIGHT
Based on the state information, the meta-level control policy prescribes the action **Call Detailed Scheduler on all lists** .

Time 36: *R1* is in state **S13** which has the following features.
F0: $< 0, 2, 0, 1 >$
F9: MEDIUM
The decommitment cost is considerable and decommitment should be avoided if possible.
F10: LOW
The current schedule has low slack and cannot fit the new tasks in the slack regions.
The meta-level control policy prescribes the following action: **Begin scheduling with the following parameters to the scheduler TSF=2, E=2, S=30%** on the two tasks in the agenda $AnalyzeRock < AT = 34, DL = 90 >$ $, ExploreTerrain < AT = 34, DL = 90 >$ and the task in execution $ExploreTerrain < AT = 15, DL = 80 >$.

Time 43: *R1* is in state **S14** when the scheduler emits the following schedule **{NegMech2, ExamineTerrain, CollectSamples, GettoRockLocation, FocusSpectrometeronRock}**. It can be noted that task $ExploreTerrain < AT = 34, DL = 90 >$ is dropped by the domain-level scheduler even though the meta-level controller had it in the scheduling list. The domain scheduler makes this decision based on it detailed computation and determines that dropping the $ExploreTerrain < AT = 34, DL = 90 >$ task and using the resources on the remaining two tasks leads to higher utility. The prescribed best action for this state is **Execute method NegMech2 in parallel with method ExamineTerrain**.

Time 46: *R1* is in state **S15** since method **NegMech2** completes successfully with utility 1. Agent's total utility is 19. Method **CollectSamples** will be enabled at time 65 The features are
F8 : LOW
Deviation from expected performance is found to be low. The chosen action is **Continue execution of ExamineTerrain.**

Time 52: *R1* is in state **S16** since method **ExamineTerrain** completes with utility 8. *R1*'s total utility is 27. The features are
F8: LOW
The chosen action is **Begin execution of GettoRockLocation**.

Time 58: *R1* is in state **S17** since method **GettoRockLocation** completes with utility 6. *R1*'s total utility is 33. The features are
F8 : LOW
The chosen action is **Begin execution of FocusSpectrometeronRock.**

Time 65: *R1* is in state **S18** since method **CollectSamples** is enabled by non-local agent. Execution of **FocusSpectrometeronRock** is interrupted and execution of **CollectSamples** begins. **FocusSpectrometeronRock** has accumulated utility of 6. Total utility is 39.

Time 77: *R1* is in state **S19** since execution of method **CollectSamples** completes with a utility/reward of 12 and task $ExploreTerrain < AT = 15, DL = 80 >$ completes with utility/reward of 24. The total reward accumulated by the system is 51. The features are
F8: LOW
This is the actual utility.
The best action is **Resume execution of FocusSpectrometeronRock**.

Time 80: *R1* is in state **S20** since execution of **FocusSpectrometeronRock** completes with utility of 12. Execution of task $AnalyzeRock < AT = 34, DL = 90 >$ completes with a reward utility/reward 18. The features are
F8: LOW
This is the actual utility.
The total reward accumulated by the system is 57. The best action is **Go to wait state**

Meta-level control reasoning in the agent leads to a cumulative utility of 57 units within the finite horizon of 100. Tasks $AnalyzeRock < AT = 1, DL = 40 >$,, $ExploreTerrain < AT = 15, DL = 80 >$, and $AnalyzeRock < AT = 34, DL = 90 >$, were completed successfully.

If the agent had used deterministic control, then the *Call Detailed Scheduler on all tasks* action would be initiated at every task arrival event independent of agent state, including methods in execution or the nearness of their deadlines. A total time of 22 units would have been spent of control actions and only two tasks $AnalyzeRock < AT = 1, DL = 40 >$, and $ExploreTerrain < AT = 15, DL = 80 >$, will complete successfully with a utility of 33 units within the finite horizon of 100.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[2] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

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

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

[5] R. Crites. Multi-agent reinforcement learning, 1994.

[6] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 49–54, Saint Paul, Minnesota, USA, 1988. AAAI Press/MIT Press.

[7] Keith Decker. Taems: A framework for environment centered analysis and design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[8] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[9] Jon Doyle. What is rational psychology? toward a modern mental philosophy. *AI Magazine*, 4(3):50–53, 1983.

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

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

[12] I. J. Good. Twenty-seven principles of rationality. In V. P. Godambe and D. A. Sprott, editors, *Foundations of statistical inference*, pages 108–141. Holt Rinehart Wilson, Toronto, 1971.

[13] Eric A. Hansen and Shlomo Zilberstein. Monitoring anytime algorithms. *SIGART Bulletin*, 7(2):28–33, 1996.

[14] 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.

[15] B. Hayes-Roth. Opportunistic control of action in intelligent agents. In *Proceedings of IEEE Transactions on Systems, Man and Cybernetics*, pages SMC–23(6):1575–1587, 1993.

[16] 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.

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

[18] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *National Conference on Artificial Intelligence of the American Association for AI (AAAI-88)*, pages 111–116, 1988.

[19] Leslie P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, 1990.

[20] M. Kinney and C. Tsatsoulis. Learning communication strategies in distributed agent environments, 1993.

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

[22] Michail G. Lagoudakis and Michael L. Littman. Reinforcement learning for algorithm selection. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, page 1081, 2000.

[23] Victor Lesser and Xiaoqin Zhang. Multi-linked negotiation in multi-agent system. *Proceedings of the First International Joint Conference on Autonomous Agents And MultiAgent Systems (AAMAS 2002)*, pages 1207–1214, 2002.

[24] Michael Littman and Justin Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CS-93-165, 1993.

[25] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.

[26] M. Mataric. Reinforcement learning in the multi-robot domain, 1997.

[27] D. J. Musliner, J. A. Hendler, A. K. Agrawala, 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.

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

[29] Yoichiro Nakakuki and Norman Sadeh. Increasing the efficiency of simulated annealing search by learning to recognize (un)promising runs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1316–1322, 1994.

[30] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

[31] M. L. Puterman. *Markov decision processes - discrete stochastic dynamic programming.Games as a Framework for Multi-Agent Reinforcement Learning.* John Wiley and Sons, Inc., New York, 1994.

[32] 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.

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

[34] Stuart Russell and Eric Wefald. Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 400–411, 1989.

[35] Paul Samuelson and William Nordhaus. Economics, 1989.

[36] T. Sandholm and R. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma, 1995.

[37] T. Sandholm and M. Nagendraprasad. Learning pursuit strategies, 1993.

[38] Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, Seattle, WA, 1994.

[39] H. Simon. From substantive to procedural rationality. pages 129–148, 1976.

[40] H. Simon and J. Kadane. Optimal problemsolving search: All-or-nothing solutions, 1974.

[41] Herbert A. Simon. *Models of Bounded Rationality, Volume 1.* The MIT Press, Cambridge, Massachusetts, 1982.

[42] 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.

[43] Toshiharu Sugawara and Victor Lesser. On-line learning of coordination plans. In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 335–345,371–377, 1993.

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

[45] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts Amherst, 1984.

[46] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[47] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[48] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[49] R. Vincent, B. Horling, and V. Lesser. An agent infrastructure to build and evaluate multi-agent systems: The java agent framework and multi-agent system simulator. In *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems.*, volume 1887. Wagner and Rana (eds.), Springer,, January 2001.

[50] 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.

[51] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge, England, 1989.

[52] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

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

[54] Shlomo Zilberstein and Stuart J. Russell. Efficient resource-bounded reasoning in AT-RALPH. In James Hendler, editor, *Proceedings of the First International Conference of Artificial Intelligence Planning Systems(AIPS 92)*, pages 260–268, College Park, Maryland, USA, 1992. Morgan Kaufmann.

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