META-LEVEL CONTROL IN MULTI-AGENT SYSTEMS

A Dissertation Presented

by

ANITA RAJA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2003

Department of Computer Science

META-LEVEL CONTROL IN MULTI-AGENT SYSTEMS

A Dissertation Presented

by

ANITA RAJA

Approved as to style and content by:

Victor R. Lesser, Chair

Shlomo Zilberstein, Member

Andrew G Barto, Member

Abhijit Deshmukh, Member

Bruce Croft, Department Chair
Department of Computer Science

ACKNOWLEDGMENTS

My first thanks go to my advisor, Victor Lesser, for giving me the guidance, support and freedom to pursue my research goals. His ability to grasp and articulate ideas and his enthusiasm are traits I will always strive for. Victor taught me innumerable lessons about research and life and I continue to be enriched by his words of wisdom. I am extremely fortunate to have been his student.

I also a owe a debt of gratitude to my thesis committee members for their patience and invaluable guidance. I am grateful for their in-depth reading and thoughtful comments of drafts of the dissertation. I thank Shlomo Zilberstein for spending numerous hours discussing my work and helping me formalize the research problem. I thank Andy Barto for his breadth of knowledge of the literature and his incredible ability to connect facets of my problem to research in machine learning. Abhijit Deshmukh helped me clarify my ideas and provided keen insights from the manufacturing systems perspective.

I thank the faculty, students and staff in the Computer Science Department for an extraordinary education. I especially thank Michele Roberts, Sharon Mallory and Pauline Hollister for countless favors they performed at short notice. My office mates, past and present, Jiaying Shen, Claudia Goldman, Sherief Abdallah, Kyle Rawlins, Andy Fast, Mark Sims, Raphen Becker, Dan Corkill, Roger Mailler and Regis Vincent made my graduate experience memorable with their lively discussions, intellectual curiosity and fun conference trips. I thank them for a providing a supportive environment.

Extensive discussions with Tom Wagner piqued my interest to study agent control and related issues. I thank Tom for mentoring me during the early years of my grad career. He deserves credit for the implementation of the DTC scheduler which was essential to the experimental studies in this dissertation. Bryan Horling got me out of a bind many times by making quick and sometimes extensive fixes to the JAF and Mass frameworks. I'm grateful for his friendship and sense of humor. I also thank Shelley Zhang for her friendship, support and kindness. Shelley deserves credit for the implementation of the negotiation protocols. I thank David Fisher for his friendship, encouragement and endless patience in reading through drafts of my papers and dissertation. I also thank my friends from First Baptist Church at Amherst and the Intervarsity Fellowship for providing a home away from home and helping me grow spiritually.

I'm indebted to a number of people for their love and generosity over the years. I thank them for keeping me grounded. Jon Aseltine helped shape my critical thinking. His friendship, support and humor have enriched my life. I'm grateful to Dagmar Jaeger for her support and for our extended discussions on many topics. She got me interested in areas outside of Computer Science including art and post-modern literature. Sergei Savin inspired me to pursue research in AI. His patience and friendship are invaluable to me. Dan Henry showed me that it is important to enjoy the simple things in life. I thank my extended family in Philadelphia for their support and encouragement. They provided the safe haven for me to escape the daily grind of graduate school.

I thank my parents, Raja Daniel and Nirmala Raja, and my sister, Poornima Raja, for their unconditional love and unwavering support. They made tremendous sacrifices so that I could follow my path and for that I'm eternally grateful. I could not have done this without them.

Finally, I give thanks to my Lord and my God for His infinite love. I consider everything a loss compared to the surpassing greatness of knowing Him.

ABSTRACT

META-LEVEL CONTROL IN MULTI-AGENT SYSTEMS

SEPTEMBER 2003

ANITA RAJA

B.S., TEMPLE UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor R. Lesser

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 without consuming too many resources in the process, is the meta-level control problem for a resource-bounded rational agent.

The focus of this research is to provide effective allocation of computation and improved performance of individual agents in a cooperative multi-agent system. This is done by approximating the ideal solution to meta-level decisions made by these agents using reinforcement learning methods. A meta-level agent control architecture for meta-level reasoning with bounded computational overhead is described. This architecture 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. The major contributions of this work are: a resource-bounded framework that supports detailed reasoning about scheduling and coordination costs; an abstract representation of the agent state which is used by hand-generated heuristic strategies to make meta-level control decisions; and a reinforcement learning based approach which automatically learns efficient meta-level control policies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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 dissertation, deliberative actions are also referred to as control actions. To do this optimally, an agent would have to know the effect of all combinations of actions ahead of time, which is intractable for any reasonably sized problem. The question of how to approximate this ideal selection and sequencing of domain and control actions without significant computational effort is the primary focus of this dissertation.

There are three classes of deliberative actions discussed in this dissertation: information gathering actions, planning/scheduling actions and coordination actions. These actions are non-trivial requiring exponential work in the number of domain actions. Sophisticated schemes that control their complexity are used in most implemented systems.

The first type of deliberative actions are information gathering actions which are of two kinds. The first kind of information gathering action 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 local processor time but they delay the meta-level deliberation process. The second kind of information gathering action 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 actions involve 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 research, planning is folded into the scheduling.

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 completion times of tasks or methods.

A problem with most multi-agent systems and agents [7, 51, 57, 38, 87] is that they do not explicitly reason about the cost of deliberative computation since they do not perform deliberative computation. 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 the overhead of computing a solution. This leads to actions that are without operational significance [69]. Taking the cost of computation into account leads to what Simon calls procedural rationality and what Good refers to as type II rationality [29]. An agent exhibits such bounded rationality if it maximizes its expected utility given its computational and other resource limits.

The intent of this research is to show that a meta-level reasoning component with bounded and small computation overhead can be constructed that significantly improves the performance of individual agents in a cooperative multi-agent system. 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 in this dissertation.

The thesis of this work is to establish the following: *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 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.*

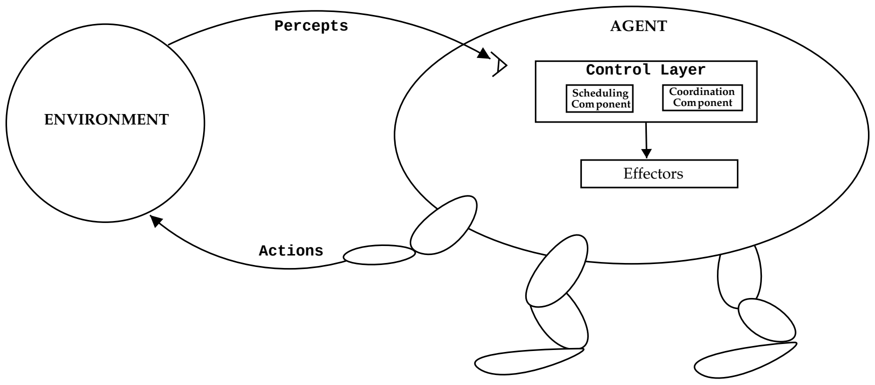


Figure 1.1. Classical architecture of a bounded rational agent

1.1 Motivation

1.1.1 Agent Decision Space Hierarchy

An agent is an entity that receives sensations from the environment and responds by performing actions that affect the environment using the effectors [58]. An ideal agent is one that always takes the action that is expected to maximize its performance criteria, given the percept sequence it has seen so far. Figure 1.1 describes the components of the classic agent architecture. In this classic architecture, when a percept is sensed by the agent, the control layer is immediately triggered regardless of the current state. The control layer determines how the percepts can be processed and mapped into domain action sequences.

Agent actions in the classic architecture are of two types: *domain actions* and *control actions*. Domain or execution actions are executable primitive actions. Sequences of these actions achieve the various tasks (goals) by affecting the environment. This research takes a simplified view of agent actions in that the only resources agents reason about are the processing resources needed to complete an action. The execution behavior of a primitive action is characterized using statistical distributions describing the amount of resources used, the likelihood of successfully completing the action and contributing to the utility of the task.

Actions taken by the control layer are called control actions. Agents are characterized by the type of control actions they take : reflexive agents use reactive control to respond immediately to percepts; goal-based agents act so that they will achieve their goals; and utility-based agents try to maximize their own rewards. The agents considered in this dissertation use a combination of reflexiveness, goal achievement and utility maximization as criteria to optimize their performance. The control layer in the classic agent architectures usually has a single option for control processing with fixed cost in terms of resource usage. This control processing will reason about the domain actions which have to be executed by the effectors.

In order to facilitate the explicit reasoning about control actions and their costs, a new category of actions called *meta-level control actions* is introduced. The classic agent architecture is augmented with meta-level control which reasons about the control actions and alternative ways of performing them. Figure 1.2 describes the new agent architecture. The arrival of percepts trigger the meta-level control layer to determine the tasks which the agent desires to pursue. The agent's control layer determines how these chosen tasks will be processed and mapped into action sequences. A more detailed description of the architecture components is provided in Chapter 3.

Control actions can be described in one of two ways: One is based on viewing a control action as an anytime algorithm that can be stopped at any point to obtain a solution. The resource usage can be controlled by determining when to stop the algorithm. Another way is to have different algorithms for performing a control action which represents trade-offs in terms of resource usage and and utility accrued. Both representations of control actions are exploited in this dissertation.

Meta-level control actions optimize the agent's performance by choosing and sequencing domain and control actions. This includes interleaving the actions such

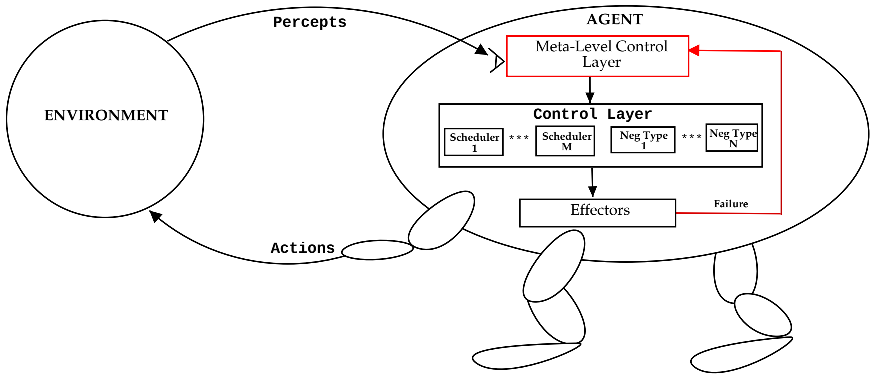


Figure 1.2. New architecture of a bounded rational agent

that tasks are achieved within their deadlines and also allocating required amount of processor and other resources to domain and control actions at the appropriate times. Meta-level control can also 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. This 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.

Figure 1.3 describes the action hierarchy space in the new augmented agent architecture for a specific event, namely *scheduling a task or set of tasks.* The three layers are the meta-level control actions, control actions and domain actions which are as described above.

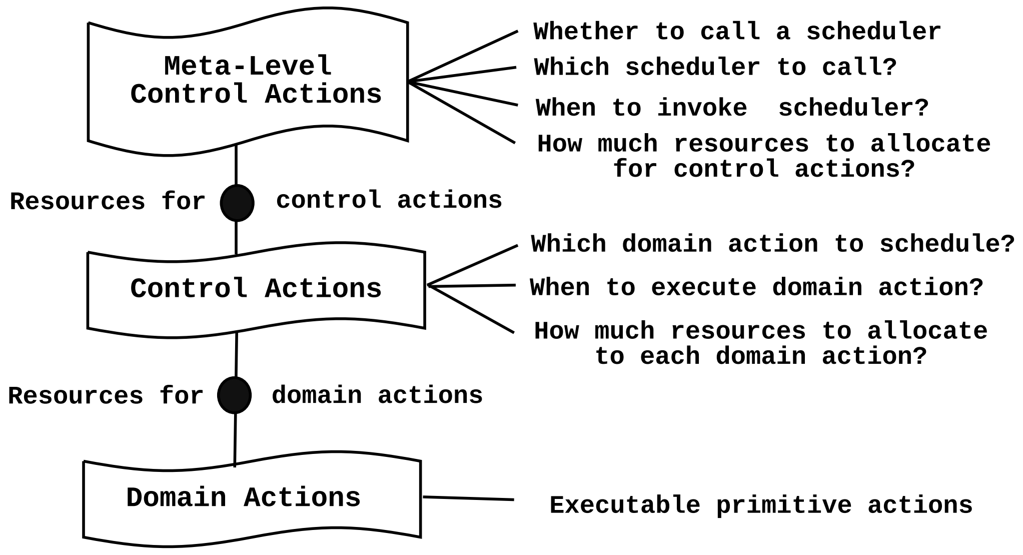

Meta-Level Control Actions
Whether to call a scheduler
Which scheduler to call?
When to invoke scheduler?
How much resources to allocate for control actions?
Resources for control actions
Control Actions
Which domain action to schedule?
When to execute domain action?
How much resources to allocate to each domain action?
Resources for domain actions
Domain Actions
Executable primitive actions

Figure 1.3. Action Space Hierarchy for Scheduling event

1.1.2 Why is this problem difficult?

There has been previous work on meta-level control [69, 60, 73, 24] but there is little that is directly related to the meta-level control problem discussed here. The difficult characteristics of this problem are :

1. complexity of the information that characterizes the state of the agent and other agents it interacts with;

2. variety of responses with differing costs and parameters available to the situation;

3. deadlines associated with these tasks;

4. high degree of uncertainty caused by the non-deterministic arrival of tasks and outcomes of primitive domain actions;

5. consequence of decisions are often not observable immediately and may have significant down-stream effects.

Russell and Wefald [61] claim that it is possible to formulate the meta-level problem using the same language as the domain-level problems, and solved using the same mechanism. The decision on how to achieve a meta-level goal is called the meta-meta-level problem. The uniformity of language enables the meta-level rules to apply to meta-meta-level goals and so on. This results in very flexible systems, but introduces the possibility of infinite regress [41, 25, 46, 23]. Like Russell and Wefald, the goal of this dissertation is not to insist on optimal control for all reasoning. Instead meta-level control actions in this research, will usually be taken without being the immediate results of extensive deliberation. Decisions to invoke the meta-level controller are hard-wired to event-based triggers. Meta-level control is implemented as a reflexive computation with low and fixed overhead. The meta-level layer can also decide to invest more resources in meta-level control itself, to gather more knowledge to assist with the decision making in case the current information is not enough.

This research takes on a much more complex version of the problem than Russell and Wefald did. The agents in this dissertation, are capable of pursuing multiple tasks (goals) concurrently. Each task is represented using the task structure and has its own deadline and potential utility that can be achieved as a result of its completion. The task structure describes one or more of the possible ways that the task could be achieved. These alternative ways are expressed as an abstraction hierarchy whose leaves are basic action instantiations. These primitive action instantiations are called domain actions. Domain actions can be scheduled and executed and are characterized by their expected quality and duration distributions. Information about task relationships that indicate how basic actions or abstract task achievement affect task characteristics (e.g. quality and time) elsewhere in the task structure and the resource consumption characteristics of tasks are also embedded in the task structure.

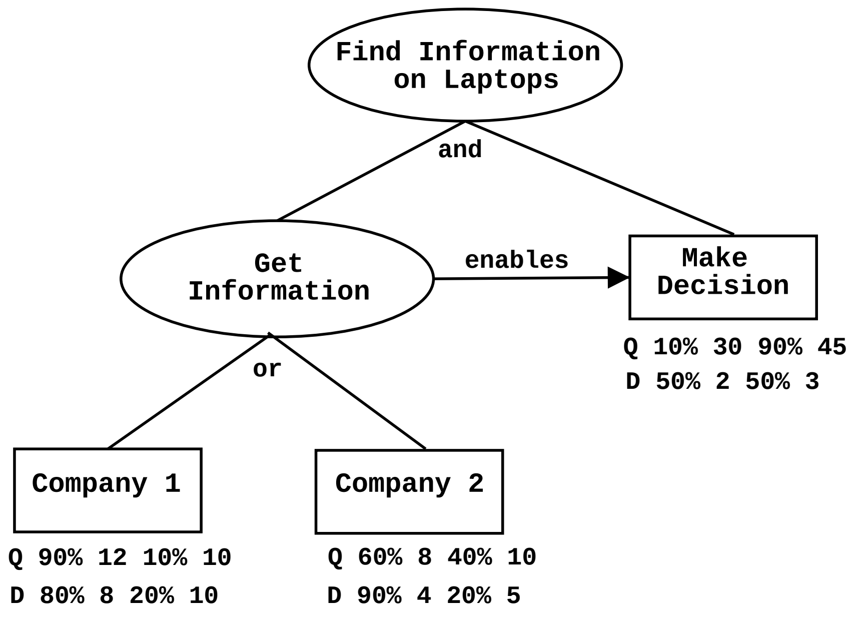


Figure 1.4. Task structure for *Find Information on Laptops* task

1.1.3 Motivating Example

Figure 1.4 describes the TÆMS task structure [19] for task *Find Information on Laptops*. A detailed description of TÆMS a domain independent framework for describing task structures is provided in Appendix A . The task is achieved by completing two intermediate steps sequentially. The first step is to *Obtain Information* - this can be done by accessing either Company 1's Site, or Company 2's Site, or both sites if time permits. Once the information is obtained, the agent will have to make a decision by choosing a laptop that fits user criteria from one of the two companies or by telling the user that none of the available choices fits her criteria. Each of the primitive actions (leaves in the task structure) are statistically characterized in two dimensions: quality and duration.

I will now describe a real-world, albeit simple example that will describe the complex interactions between the domain and control actions of the agent. Consider an administrative agent, *Yogi*, capable of performing multiple tasks such as *find information on laptops with the best value*, *answering the telephone* and *paying bills*. Suppose

Yogi receives these tasks dynamically and does not know the arrival model ahead of time. Each of the tasks have an associated task structure. In the case of the *Find Information on Laptops* task, the domain actions are **Company 1 Site** and **Company 2 Site** and **Make Decision**. These are actions that affect the environment. Each task has a set of domain actions as described by its task structure.

The control actions for *Yogi* in this scenario are *scheduling the task with varying levels of effort*. Scheduling the task involves finding the appropriate sequence of domain actions given the time and other resource bounds. Examples of control actions for the *Find Information on Laptops* task would be whether to search

1. both Company 1's site and Company 2's site
2. either Company 1 or Company 2
3. none of the sites and to return a null answer (due to lack of resources)

The meta-level control actions are *whether to do a task* and *how much time and effort to spend on control actions related to the task* and *how much time and effort to spend on domain actions related to the task*. Figure 1.5 describes some of the meta-level control decisions *Yogi* has to make.

Suppose *Yogi* does not perform any meta-level reasoning about the importance or urgency of the tasks. *Yogi* will then spend the same amount of time deciding whether to answer a ringing phone as it does on deciding which laptop manufacturer sites to visit. If *Yogi* is equipped with meta-level reasoning capabilities, it will recognize the need to make quicker decisions about the phone call than about the laptops since there is a tight constraint on the ringing phone, namely that the caller could hang up.

Meta-level control will also allow *Yogi* to dynamically change its decisions based on its current state. For instance, if the deadline for determining the laptop information

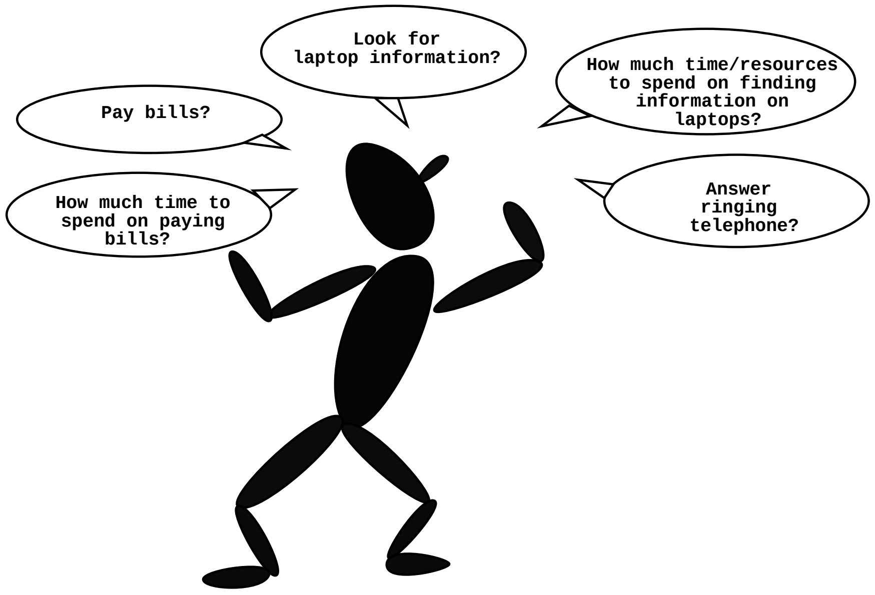

Pay bills?
Look for laptop information?
How much time/resources to spend on finding information on laptops?
How much time to spend on paying bills?
Answer ringing telephone?

Figure 1.5. Meta-Level Control Decisions to be taken by *Yogi*

is imminent, *Yogi* could decide not to answer any phone calls until the search is completed and the suggestion is made to the user.

Another example is as follows: suppose *Yogi* is expecting an important phone call in the next 5 minutes and expects the call to last an hour. Now suppose *Yogi* gets a request to find a laptop given a set of criteria within 20 minutes and the utility of this task was lower than that of engaging in the important phone call. If *Yogi* accepts a task and then renegs on the commitment, it will be charged with a significant penalty. If *Yogi* refuses to do the laptop tasks, another agent would be assigned the task and it could get done in time. Also suppose *Yogi* has some bills that it has to pay and paying bills is an interruptible task with the deadlines being far out in the future. *Yogi*, in this case, should decide to refuse the laptop task and choose to pay bills while waiting for the phone call. When the phone call ends, *Yogi* can resume paying bills as long it is does not have any new tasks in the agenda to reason about.

Yogi is thus able to make better decisions about answering calls as well as completing its other tasks by dynamically adjusting its decision based on its current state and the tasks at hand.

1.1.4 Taxonomy of meta-level decisions

There are five types of event triggers that require meta-level decision making

1. Arrival of a new task from the 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. Decision to schedule a new set of tasks or to reschedule existing tasks
5. Significant deviation of online schedule performance from expected performance

The meta-level controller is invoked when one of the five triggers occurs. The choice of these particular trigger events was deliberate because in the task allocation domain and other similar domains, these events occur frequently through out the finite horizon under consideration.

The meta-level controller was specifically defined as a trigger-based mechanism and not as a component invoked at uniform time intervals for the following reasons: Periodic activation can be too frequent in loosely constrained environments leading to unnecessary overhead of meta-level control and can be too infrequent and hence ineffective in very dynamic environments. The trigger based mechanism is a general solution which can handle the entire spectrum of environments without any tweaking. Secondly, a trigger-based mechanism is suitable for domains where activities requiring meta-level control do not necessarily arrive in a uniform distribution. If the activities tend to arrive in bursts, then the periodic activation method would be too infrequent during the burst periods and waster resources during the periods of inactivity.

An example of an event trigger that wasn't included is a situation in which one of the primitive actions goes out of control and continues execution beyond its expected finish time resulting in overuse of allocated resources. Although this event can lead to bad performance, it was not added as a trigger for two reasons: one that it occurs infrequently and secondly, since the other triggers occur quite frequently, the meta-level controller will be invoked sooner than later and the meta-level controller will recognize the overuse of resources and abort the errant primitive action.

The following are some characteristics of problem domains where the type meta-level control described in this dissertation may provide improvement in performance: The agent internally generates a new goal as a result of sensing perceptions. These goals can be revised as a result of sensing internal actions and their effects on the environment. Goal analysis, performed by the control layer, is the process by which an agent receives a goal or set of goals as input and outputs a sequence of executable methods with start time and finish constraints and associated expected utility. Meta-level control is process of deciding among the following choices: to drop the goal and not do any analysis; to delay goal analysis; reason about the amount of effort to go into goal analysis; and to determine the context of the goal analysis - whether to analyze it a single goal or multiple goals within a single agent perspective; or to analyze single or multiple goals in the context of a facilitating agent's goals.

Meta-level control is useful in situations where options for goal analysis are expensive in other words the costs or accumulated costs affect agent performance detrimentally. Meta-level control is also useful when the cost of goal analysis is significantly more expensive than cost of meta-level control actions. It is also useful where a choice has to be made about the type of goal analysis and the options for goal analysis have significantly different costs and produce results with significantly different utilities.

The range of the number of options for goal analysis, the difference in the performance characteristics where meta-level control will be most effective will be the subject of future research.

In some situations, meta-level control can also be viewed as an effective and inexpensive alternative for making the same decisions as the more expensive goal analysis. For instance, when a goal arrives and its deadline is too close for the agent to execute the goal, the meta-level controller will quickly and inexpensively determine that the goal has to be dropped. The goal analysis would have come up with the same decision after completing its computations which have associated costs. In the absence of meta-level control, these costs can add up leading to degradation in performance.

1.1.5 Cost-Benefit Analysis of Meta-Level Control

In order to provide a clear picture of these five decisions described above, consider the simple scenario consisting of two rovers *Fred* and *Barney*. Rovers are unmanned vehicles equipped with cameras and a variety of scientific sensors for the purpose of planetary surface exploration. The discussion here will specifically focus on the various meta-level questions that will have to be addressed by *Fred*. Figure 1.6 describes *Analyze Rock*, also called task *T0*, and *Explore Terrain*, also called task *T1*, which are the tasks performed by *Fred*.

In this example, each top-level task is decomposed into two executable primitive actions. In order to achieve the task *Analyze Rock* , *Fred* 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

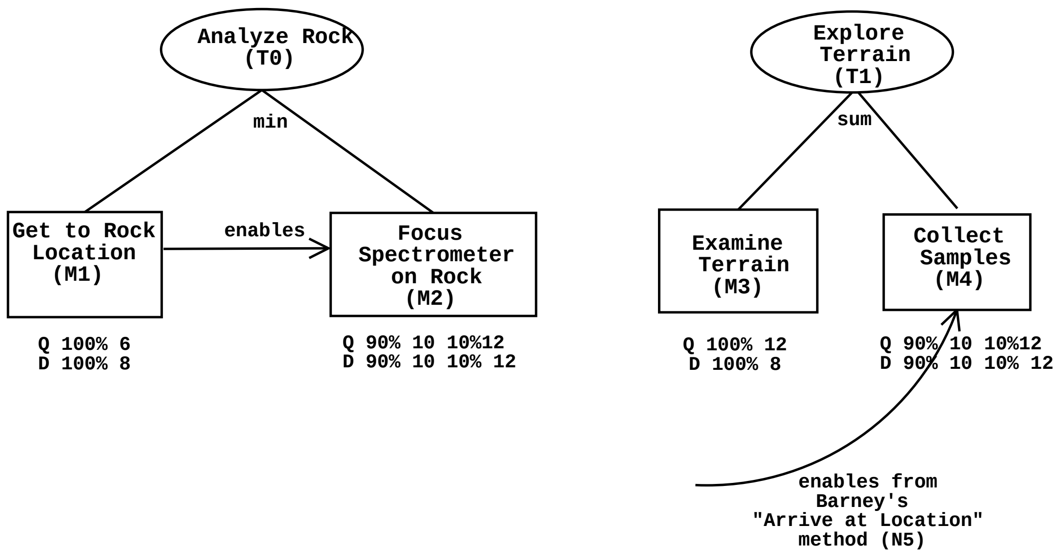


Figure 1.6. Tasks that can be performed by agent *Fred*

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*, *Fred* 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).

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

The following are some of the specific meta-level questions that will be addressed by any individual agent. Each meta-level question is followed by a description of

Fred'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 *Fred* schedule newly arriving task *Tx* 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 and its deadline is very close and high probability of high utility task arriving in the future, then it should be discarded. This means *Fred* chooses not expend its limited resources on a low priority task and instead will wait for future high priority task. If the incoming task *Tx* has very high priority, in other words, the expected task utility is very high and it has a relatively close deadline, then *Fred* 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: If the new task is scheduled immediately, the scheduling action costs time, and there are associated costs of dropping established commitments if the previous schedule is significantly revised or completely dropped. These costs are diminished or avoided completely if the task reasoning 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 *Barney's* method *Arrive at Location*, be included in the *Fred's* schedule?

Benefit: If method *Collect Samples* is included in *Fred's* schedule, *Fred* can increase its total utility.

Cost: This implies that *Fred* and *Barney* have to negotiate over the completion time of *Barney's* method *Arrive at Location* and this will take time.

(b) **Meta-Level Question:** If *Fred* decides to negotiate, it should also decide whether to negotiate by means of a single step or a multi-step protocol [45] 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 a single shot protocol which is quick but has a chance of failure be used or a more complex protocol which takes more time and has a higher chance of success.

Benefit: If *Fred* 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 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 *Fred* and *Barney* using a particular negotiation protocol fails, should *Fred* retry the negotiation with *Barney* again [1]; if so, should *Fred* use the same negotiation mechanism as before or

[1]An interesting extension to this meta-level question which has not been explored in this thesis is whether *Fred* should renegotiate with *Barney* or with another agent *Wilma* who is capable of performing the same task. The states and hence the *MetaNeg* information of 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 *Fred* to negotiate with *Wilma* rather than *Barney*

an alternate mechanism; and how many such retries should take place?

Benefit: Negotiation is preferred if *Fred* will receive high utility as a result of Barney 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: This implies that *Fred* and *Barney* have to negotiate over the completion time of Barney's method *Arrive at Location* and 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 *Fred'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 without having to bear the overhead of a reschedule.

   Cost: If the schedule performance is not as expected, then rescheduling has to be called anyways and *Fred* 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 towards 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 *Fred's* schedule deviates from expected performance by threshold α, should a reschedule be invoked automatically?

   Benefit: If *Fred* 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: There is a cost associated with calling the scheduler and revising the commitments from the previous schedule.

1.2 Assumptions

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 the completion of a goal derives quality 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 the horizon is set to 500 time clicks in the experiments in this dissertation, 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). Future work would involve extending the work to domains with finite and infinite horizons.

The high-level goals are generated in one of two ways: internal or external events are sensed; and requests from other agents for assistance. These goals must often be completed by a certain time in order to achieve any quality. It is not necessary for all high-level goals to be completed in order for an agent to derive quality from its activities. The partial satisfaction of a high-level goal is sometimes permissible while trading-off the amount of quality derived for decrease in resource usage.

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. Coordination decisions 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, in this case it is negotiation for task allocation, are binding and assume that other agents will not decommit from their commitments at later stages.

1.3 Formal Model

This dissertation describes a heuristic approach rather than a theoretical approach to meta-level control reasoning in multi-agent systems. However, it is possible to define a decision theoretic formulation of the meta-level control problem.

1. Let S be the set of states of the agent and $s_i \epsilon\ S$ denote the agent state at stage i, $i = 0, 1, 2, 3 \ldots, n$

2. A is the set of possible control actions and $a_i \epsilon\ A$ 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 task type, task arrival rate and task deadline tightness. 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. s_j is the new state reached when the agent is interrupted by an event requiring meta-level control reasoning while executing control action $\pi(s_i)$ followed by the execution of corresponding domain actions that follow $\pi(s_i)$.

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

   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.

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

7. $P(s_j|s_i, \pi(s_i))$ is the probability that agent is in state s_j as a result of taking action $\pi(s_i)$ which is the action prescribed by policy π in state s_i.

The above model defines a finite Markov decision process [3].

According to decision theory, an optimal action is one which maximizes the agent's expected utility, given by

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

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

which can be computed as follows

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

The meta-level control problem 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 [3] and reinforcement learning [75] methods. These methods will implicitly determine the transition probability model and reward function defined previously.

In this work, the complexity of the state space makes it difficult to find the optimal policy. So an approximate meta-level control policy is found using a abstract state representation which will capture only the information relevant to the decision making process.

1.4 Example Application Domains

The following are some real-world examples where effective meta-level control can lead to improved system performance.

Planetary Rovers: Rovers are automated vehicles used in planetary exploration. A rover has a set of pre-assigned tasks. New tasks can be generated when certain environmental phenomenon are observed. The domain actions are the external actions the rover takes on the environment to accomplish its tasks. The control actions are the computational actions the rover has to take to determine which domain actions to execute and how much resources to allocate to each domain action. The rover must make situation-specific run-time meta-level decisions on which tasks to perform and how much effort to put into deciding how to best accomplish each task given the current context. The rover also has to make meta-level decisions on when to make autonomous choices and when it is necessary to initiate communication to its home base to receive directions. Communication is expensive, however, the decision choices provided by home base are as good as, if not superior, to locally made choices. If it is overloaded with tasks or anticipates being overloaded with tasks in the future, the rover must also make meta-level choices on whether to negotiate with another rover to transfer a task. This type of meta-level control will allow the system to maximize the utility gained from a set of tasks and to revise the task allocation dynamically in response to changing circumstances.

Truck Scheduling: The task of a dispatch agent in a freight forwarding company is to schedule a fleet of vehicles and their drivers such that all incoming and accepted transportation orders will be performed with minimal costs and maximal profit. The domain actions are actions on the environment which involve taking the vehicles out on the road and making the deliveries in order to complete the delivery contracts. The control decisions involve determining the route for the trip, the sequence of the deliveries such that deadlines are met and limits on driving hours are obeyed. The agent must make meta-level decisions on which delivery contracts to accept and which to reject based on its current contracts and resource constraints. The agent may also decide to accept a high utility contract, even if it is over committed and can't complete the job itself. The agent will make the decision to accept the contract based on a successful negotiation with a sub-contractor to complete the task and it will do this if the task has very high utility or if the agent wants to maintain a good history with the valued customer. The resource constraints in this domain are the number of available trucks, their prescheduled routes, space available and driver allocation. The delivery contracts will have constraints on time windows for pick up and delivery which have to be adhered to.

Financial Portfolio Management: A portfolio manager agent as described in Decker et al [16] has the task of managing an investment portfolio and achieving a specified rate of return over time, using various information sources. This task of monitoring portfolio assets involves processing the enormous amount of continually changing information. The domain actions involve accessing of the various information sources about assets and processing the information that is obtained. The control actions involve dynamically sequencing the information gathering and information processing actions such that decisions on assets can be made before the information gathered becomes irrelevant. The meta-level actions involve determining which of the variety of assets the agent should target; and choosing the kinds of information sources (market data, financial report data, technical models, analyst reports, breaking news) to consider while making a decision on each asset and which ones to definitely avoid.

Team Selection: Consider an agent whose role is to be a basketball coach. Suppose the agent is aware of the strengths, weaknesses and current physical condition of each player on the team. The agent also knows the characteristic features of the opposing team. Based on this knowledge, the coach agent has to make meta-level decisions on which players should be benched, which players to have as back-ups and which of them should definitely be on the court. Once the meta-level choices are made, the control decisions would involve devising a strategy based on the abilities of chosen players to win the game; this could include decisions on how much game time to allocate to each of the chosen players and when to have them on the court. The domain actions would be the actions which happen on the court during the play.

1.5 Overview of Contributions

1. **Meta-level Control Architecture:** I present a novel meta-level control agent architecture for bounded-rational agents operating in a complex multi-agent system. The architecture allows for accounting for costs at all levels of reasoning. The meta-level control mechanism has limited and bounded computational overhead and supports reasoning about planning and scheduling costs as first-class entities. The system adapts its computational effort from exact to approximate computation based on its resource constraints.

2. **Application of Empirical Reinforcement Learning to a Complex Domain:** I show that Reinforcement learning is a viable approach for studying the meta-level control problem. I model the meta-level controller using a Markov decision process such that an Q-Learning algorithm described in [72] can be applied to learn efficient meta-level control policies. I also describe experiments on learning curve saturations, policy generalization and issues faced by co-learning agents

3. **Abstraction-based Reasoning:** I show that meta-level reasoning based on abstractions of the real system state makes it computationally feasible. A few high-level features that accurately capture the state information and task arrival model enable the meta-level control component to make useful decisions which significantly improve agent performance.

   Another form of abstraction-based reasoning involves representing tasks at varying levels of detail. Knowledge about all tasks is gathered manually through an offline process. This knowledge includes potential schedules in the form of linear sequences of primitive actions and their associated performance characteristics such as expected quality distribution, expected duration distribution, and expected duration uncertainty for achieving the high level tasks. The expected values are acquired by systematically searching over the space of objective criteria. The abstraction hides the details of these schedules and provides only the high level information necessary to make meta-level choices.

4. **Adjustable Autonomy:** Adjustable autonomy allows systems to operate with dynamically varying levels of independence, intelligence, and control. A human user, another system, or the autonomous system itself may adjust an agent's "level of autonomy" as required by the current situation. Most current work deals with the ability of humans to adjust the autonomy of agents in multi-agent systems[65]. This work uses the notion of agent-centered autonomy which is closely related to idea of building functionally accurate cooperative distributed systems [42]. The FA/C work describes locally autonomous agents, which plan independently but act as part of a larger system and exhibit diverse behavior. These agents may be fully cooperative or choose to act more selfishly. Control actions in the system described in this thesis such as scheduling and coordination require resources and hence reduce the local autonomy of agents to perform local domain activities. I also show how meta-level control can be viewed as a decision process which determines the local autonomy of an agent.

5. **Comparison study of various meta-level control strategies** I compare the performance of the learning-based meta-level control methodology to increasingly sophisticated approaches, all based on the high-level features used to represent system state, to handle the meta-level control problem. In the most simple case, the meta-level control policy is a deterministic policy where the

control action that has the highest expected utility and is most expensive is always chosen independent of the agent's state. In this case, there is no explicit meta-level control and reasoning about control costs is ignored. The second case is where the meta-level controller randomly chooses the control action from the set of possible choices. and high quality control action is always chosen. In the third case, the meta-level control policy is a heuristic policy is a set of hand-generated rules that are mostly environment independent. Then, I explore a set of more sophisticated set of hand generated rules that use knowledge about task characteristics including arrival times and deadlines. Finally, a reinforcement learning approach which uses the high-level state features to automatically learn the meta-level control policy is evaluated.

1.6 Outline

Chapter 2 situates this research in the context of extensive research done in the fields of planning, control, meta-level control and machine learning.

Chapters 3, 4 and 5 comprise the body of the dissertation. They describe the specifics of the meta-level control architecture, and a succession of methodologies for constructing meta-level control policies.

Chapter 3 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. This is followed by a formal description of the problem. A detailed example illustrating the functionality of the infrastructure is presented.

Chapter 4 describes 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

Chapter 5 begins with a discussion about why reinforcement learning is an appropriate solution approach for meta-level control. Then, the states and actions of the underlying Markov decision process [56] used by the learning component are described. Experiments describing the viability of an empirical reinforcement learning algorithm both in single agent as well co-learning multi agent environments are presented.

Chapters 4 and 5 also present empirical results for single-agent and multi-agent scenarios along with the strategy.

The dissertation concludes with Chapter 6, in which the strengths and limitations of this research and directions for future research are discussed.

CHAPTER 2
RELATED WORK

2.1 Agent Control and Architectures

There has been enormous amount of work on intelligent agent control e.g. [4, 26, 50, 82, 89]. 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 in my research reasons explicitly about control costs and includes reasoning about costs at all levels of computation.

2.1.1 Subsumption and three-layer architectures

Reactive planning, which uses pre-defined sensor-action rules, can effectively respond to dynamic changes in real-time environments. However, it is in general challenging to strategically reason about objectives using reactive planning because it does not account for unexpected developments in the environment. Therefore, ideally, deliberative and reactive planning should be integrated in environments with unknown dynamics.

There has also been a lot of work on layered architectures within reactive planning. In the mid-1980's, Brooks introduced the Subsumption Architecture [10] which uses a layered finite state machine to represent the agent function and stresses the use of minimal state information. The subsumption architecture is not equipped to handle multiple tasks since it cannot switch between control methods automatically. Three different groups of researchers [12, 27, 5] working more or less independently came up with a three layer architecture that addresses this control problem at about the same time. The three layer architecture arises from the empirical observation that effective algorithms for controlling mobile robots tend to fall into three distinct categories : a reactive feedback control mechanism, a reactive plan execution mechanism, and a mechanism for performing time consuming deliberative computations. The three layer architecture is based on the premise that algorithms of the higher layer can provide effective computational abstractions to the layer just below it. The three layers are: the deliberative planning layer, that does detailed reasoning about goals and resource constraints; the skills layer, that is based on a reactive feedback control mechanism that does not use state information; and a sequencer layer, that connects the other two layers and governs routine sequences of activity that rely on internal state. The meta-level agent architecture described in this dissertation also has this view that higher levels of the architecture provide useful abstractions to reason about the lower levels. The 3T (3Tier) [6] architecture is constructed to allow for integration of control algorithms in the bottom layer with advanced planning and scheduling components in the top layer. The top layer is event driven and computationally expensive. The 3T architecture, unlike the meta-level architecture, does not have a meta-level layer that explicitly reasons about deliberative costs. Also, in the meta-level architecture, only two of the three layers are decision making layers and both these layers use internal state to make their decisions.

2.1.2 Beliefs-Desires-Intentions Architectures

The Beliefs-Desires-Intentions (BDI) model [8] is probably the most popular approach towards the design of intelligent agents. The model defines trigger behaviors driven by conceptually modeled interactions and goals rather than procedural information. In addition, the BDI model seems to be a functional abstraction for the higher-level reasoning processes such as action selection. Intelligent Resource-bounded Machine Architecture (IRMA) [9], an architecture for resource-bounded [1] deliberative agents, has four key symbolic data structures: a plan library, and explicit representations of beliefs, desires, and intentions. Additionally, the architecture has: a reasoner, for reasoning about the world; a means-ends analyzer, for determining which plans might be used to achieve the agent's intentions; an opportunity analyzer, which monitors the environment in order to determine further options for the agent; a filtering process; and a deliberation process. The filtering process is responsible for determining the subset of the agent's actions that are consistent with the agent's current intentions. The choice between competing options is made by the deliberation process. These features are very similar to my meta-level control architecture. One drawback with BDI architectures is that they don't adapt well to dynamic environments which are characterized by unknown task arrival models. Also, BDI architectures do not reason explicitly about the cost of the means-end analysis to determine the agent's intentions. The meta-level control architecture has learning capabilities which support reasoning about control activities and reasons about the costs of computations at all levels.

2.1.3 Procedural Reasoning System

The Procedural Reasoning System (PRS) [28] 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

[1]The agent resource considered is computational power

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 my research. 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.

2.1.4 Guardian

Hayes-Roth [32] 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 dissertation similarly computes approximate solutions rather than optimal solutions.

The system she develops that solves problems closest to the complexity to the problems I am interested in is Guardian [33]. 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.

2.2 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 [22] 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 [34] 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 [69]. Simon's work has addressed the implications of bounded rationality in the areas of psychology, economics and artificial intelligence [71]. 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 [70] 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 [29] 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.

Russell, Subramanian and Parr [60] cast the problem of creating resource-bounded rational agents as a search for the best program that an agent can execute. This definition of rationality does not depend on the method used to create a program or the method it uses to do computation but only on the behaviors that result from running the program. In searching the space of programs, the agents can be optimal for a given class of programs or they can approach optimal performance with learning, again given a limited class of possible programs. The assumption of a perfectly rational agent is not feasible in real-time multi-agent systems. The approach I take in my work is a constructive one that Russell calls meta-level rationality. By approximating the correct meta-level decisions, the agents attempt to produce high expected utility within the resource limits. However, the agents provide no guarantees about their optimality.

Flexibility in manufacturing systems is a feature which has a functionality similar to meta-level control. Flexibility is widely understood as the ability of a manufacturing system to respond to change [14], often extended with the proviso that response should be rapid and cost effective. From a managerial point of view flexibility combines, in an often uneasy way, the element of strategy with that of dealing with uncertainties in the configurations of demand requiring rapid adaptation of resources [67]. From the classifications given in the literature a number of common themes emerge: Flexibility is the ability to deal with variability and uncertainty; the sources of variability and uncertainty can be either planned or unplanned and due to internal or external events; the ability to respond or be flexible should be measured in terms of range and speed of response. They also study the costs and benefits of flexibility and consider the utility of using flexibility to overcome weakness which could be addressed in other ways.

Uncertainty is reasoned about indirectly in the meta-level control work described in this dissertation. The complex scheduler reasons about uncertainty of domain actions explicitly. Also the slack parameter which determines the tightness of schedules allows foe unexpected events. Internal uncertainty can be improved by studying the performance characteristics of the control options in more detail. The uncertainty in peer agent behavior is also reduced by negotiation with the other agents.

2.2.1 Planning

Meta-level control has also been called meta-level planning. As this term implies, an agent can plan not only the physical actions that it will take but also the computational actions that it will take. On-line meta-level control uses computation to explicitly decide which object level computations to perform. The central questions are the types of decisions to be made and the algorithm used for making each decision. For planning, the decisions arise from the choice points in non-deterministic planning algorithms, and from deciding when to begin execution. Meta-level control algorithms can be simple heuristics or a recursive application of the full planning algorithm.

Stefik's Molgen planner [73] uses the base level planner to create meta-level plans. Molgen considers two levels of meta-level planning, in addition to base-level planning. The actions at each of these meta-levels create plans for the next lower level. Molgen does not reason about resource usage. In contrast, my approach uses only a single layer of meta-level control, uses algorithms and heuristics tailored to making particular meta-level control decisions on resource trade-offs. Additional layers of meta-level control have a diminishing rate of return since each layer adds additional overhead and there is a bound on how much meta-level control can improve performance.

2.2.2 Decision Theory

Decision theory provides a measure of an agent's performance that the meta-level controller can use when making meta-level control decisions. Russell and Wefald apply decision-theory and meta-level control to standard search problems. Their DTA* algorithm [59] uses estimates of the expected cost and expected gain in utility for possible computations to decide which computation to perform or whether to act. The algorithm is myopic and considers only the implications for the next action to be executed. Their method for deciding which node in the search tree to expand can be cast in terms of sensitivity analysis. The sensitivity analysis only considers the effect of changing one variable at a time. Their work is defined in the context of single agents and the meta-level decisions are independent and do not affect downstream decisions. This dissertation defines a sequential decision making problem for multi-agent environments. The DTA* algorithm only considers the effect that a computation will have on the value of the next action, while I consider the effect on the value of an entire plan. The focus on plans rather than individual actions is appropriate in domains where a sequence of actions is required to achieve a task and the value of an action depends on the actions that will follow it.

2.2.3 Anytime Algorithms

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 [15]. 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 [52]. 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 [88] 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 [89], can be used to compile programs consisting of both anytime and traditional algorithms [2]

Hansen and Zilberstein [30] 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 dissertation. 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.

The meta-level control approach differs from the work on monitoring anytime algorithms in that it deals with many different types of meta-level decisions that interact that each other. This complicates the reasoning process. The environment is also characterized by uncertainty in task arrivals, execution durations and utility distributions. Furthermore, the multi-agent aspect of my work adds another dimension of complexity to the problem.

Harada and Russell [31] 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[3]. 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

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

[3]Personal communication with second author.

effects. The domain is characterized by uncertainty in action durations and utility accrued. The partial global planning [24] approach is a flexible framework for coordination where nodes can balance their needs for predictability and responsiveness differently for different situations. In this framework, nodes exchange information about their tentative local plans and develop partial global plans (PGPs) to represent the combined activities of some part of the network that is developing a more global solution. To dampen their reactions to deviations, nodes need to know when deviations are negligible and should be ignored. The PGPlanner considers a deviation between actual and predicted times to be negligible if that difference is no larger than the time-cushion. The time-cushion is a user-specified parameter that represents negligible time and balances predictability and responsiveness. My learning based approach allows the agent to adjust its response dynamically based on its current state.

COLLAGE [55] is a learning system that uses to meta-level information to learn to choose the appropriate coordination strategy from among a class of strategies. They provide empirical evidence for the benefits of learning situation-specific coordination. Kuwabara [38] proposes a meta-level control mechanism for coordination protocols in a multi-agent system. AgenTalk [39], a coordination protocol description language, is extended to include primitives for the meta-level control. The meta-level control mechanism allows agents to detect and handle unexpected situations by switching between coordination protocols. These two systems choose a situation-specific strategy from a number of options. However, they do not deal with the notion of bounded rationality and do not account for the cost of meta-level control. They also limit their work to coordination protocols and don't consider other control activities.

2.2.4 Reinforcement Learning

Algorithms for sequential Reinforcement Learning (RL) tasks have been studied mainly within a single agent context [1, 77, 84]. Some of the later work described below have applied RL methods such as Qlearning to multiagent settings. In many of these studies, the agents learn about either simple dependent tasks or independent tasks. Sen et al. [66] describe 2-agent block pushing experiments, where the agents try to make the block follow a line by independently applying forces to it. Tan [79] reports on grid-world predator-prey experiments with multiagent RL, focusing on the sharing of sensory information, policies, and experience among the agents. Unfortunately, just slightly harder predator-prey problems [64] and prisoner's dilemma [63] 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.

Crites and Barto [13] propose applying multiagent RL algorithms to elevator dispatching, where each elevator car would be 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 [47] 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 the research described in this dissertation, the entire Q-function, not just a single entry, is stored by each agent. Littman [48] 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.

Weiss [85] presents Bucket Brigade based sequential RL experiments in a simple blocks world problem. The cooperative agents with partial views share a goal but do not know what the goal is. Tesauro [80] has successfully applied RL to Backgammon. Shoham and Tenneholtz [68] describe a simple learning algorithm called Cumulative Best Response that performs well in identical payoff settings but performs poorly in the IPD. Despite some weak theoretical guarantees of eventual cooperation, in practice, agents using this learning rule usually fail to reach cooperation in hundreds of thousands of iterations.

Lagoudakis and Littman [40] 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 this dissertation complicates the reasoning process. Other multiagent learning research has used purely heuristic algorithms for complex real-world problems such as learning coordination strategies [74] and communication strategies [37] with varying success.

The meta-level control architecture described in this dissertation differs from the above mentioned works in that it uses RL to make meta-level control decisions in a complex sequential decision making, cooperative multiagent 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 [78] 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 [54] developed an approach to RL in which the policies considered are constrained by 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 [21] 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 and this dissertation work 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.

CHAPTER 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, the 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.

3.1 Agent Architecture

The open agent architecture described in Figure 1.2 provides efficient meta-level control for bounded rational agents. In this section, a detailed description of the architectural components and their interactions is provided.

The components of the architecture are: the environment, the meta-level control layer, control layer and the effector layer. The control layer consists of the different schedulers and negotiation components. The effector layer is the execution subsystem.

Figure 3.1 shows the control flow within the architecture. In this complex architecture, the control components such as the schedulers, negotiation components and execution subsystem interact with the meta-level control component. 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 environment consists of a task generator which generates tasks for individual agents based on an arrival model. 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 or effector layer 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 uses the pre-computed abstract information (aka task abstraction) about the task to select the appropriate schedule which fits the criteria. A task abstraction in this dissertation is a data structure which captures the information on the alternative ways of achieving the associated task. Task abstractions support reactive control for highly time-constrained situations. When an agent has to schedule a task but doesn't have the resources or

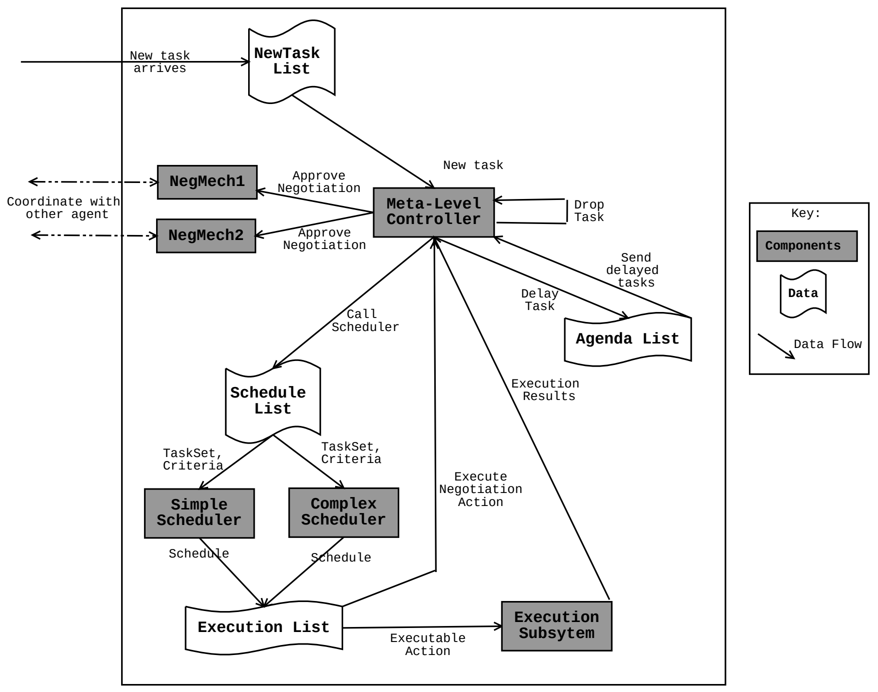


Figure 3.1. Control Flow in Meta-Level Control Agent Architecture

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. These performance characteristics are discovered by systematically searching over the space of objective criteria. Examples of schedules and their performance characteristics are provided in Appendix A. The task abstraction hides the details of these schedules and provides only the high level information necessary to make meta-level choices. Figure A.3 describes examples of alternative schedules produced by varying the objective criteria.

Complex Scheduler: The domain level scheduler depicted in the architecture is an extended version of the Design-to-Criteria (DTC) scheduler [82]. A detailed description of DTC and its schedules is given in Appendix A. 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. A detailed description of the scheduler parameters are provided later on in this section.

Adjustable autonomy is the capability of agents to dynamically vary their own autonomy, by agreeing to perform tasks and also by transferring decision making control to other agents as a result of coordination. Although the meta-level control architecture described in this chapter was not constructed with adjustable autonomy as an explicit first-class objective, it is a valuable feature that emerges as a consequence of the resource-bounded nature of the problem being solved.

Negotiation Protocols: There are two types of negotiation protocols [45]: **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. The complexity of the negotiation protocol required is proportional to the value of the negotiated task and availability of slack. The availability of slack in the other agent is a good (but not always) reliable indicator of whether the requested task can be accommodated in the agent's schedule. 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 in terms of the delays on when a negotiated task can start and also the overheads for doing the computation and communication involved in the negotiation but has a higher probability of success.

The Meta-Level Control (MLC) Layer is invoked when certain exogenous or internal events occur. The controller computes the corresponding abstracted agent state and determines the best action prescribed by the policy for that particular task environment. There are five different approaches to determining the meta-level control policy: the simplest policy is a random policy, where a meta-level control action is chosen at random independent of the current context; a deterministic policy which always chooses the highest quality and possibly most expensive meta-level control action independent of context; a simple hand-generated heuristic policy which dynamically adjusts to context although its reasoning is myopic in the case of the naive heuristic strategy (NHS); a more complex heuristic policy based on task arrival information in the case of the sophisticated heuristic strategy (SHS); and a automatically learned policy based on a reinforcement-based learning strategy (RLS).

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. A brief description of the reasoning process was provided in Chapter 1. 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. A description of how meta-level control allows for adjustable autonomy implicitly is also provided with each of the five events described below.

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 3.2. 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]. The agent, in deciding not to commit to a new task immediately, is implicitly choosing to retain its current level of autonomy in terms of available resources. 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 agent, in the case of actions A2, A3 and A4, implicitly determines that it is more valuable to give up varying amounts of its autonomy (depending on the priority of the new task)and commit to processing the new task. The meta-level controller might decide that it does not have enough information to make a good decision and will consequently choose to spend more time in collecting features which will help with the decision making process [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). [1]

To elucidate this control process, instances of the state of agent *Fred* (described in Chapter 1) and the corresponding decision choice made by the meta-level controller are provided.

An example of the above described decision process occurs when the *Fred* is in State *S1*. It represents the situation at time 2. *Fred* 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. *Fred's* meta-level controller is invoked. All the other lists are empty and *Fred* has not executed any task and has accrued zero utility. Based on its current state, *Fred's* meta-level control decision is to *Call the Detailed Scheduler.*

State S1:

CurrentTime : 2

[1]The cost of computing complex features for the experiments described in this dissertation 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

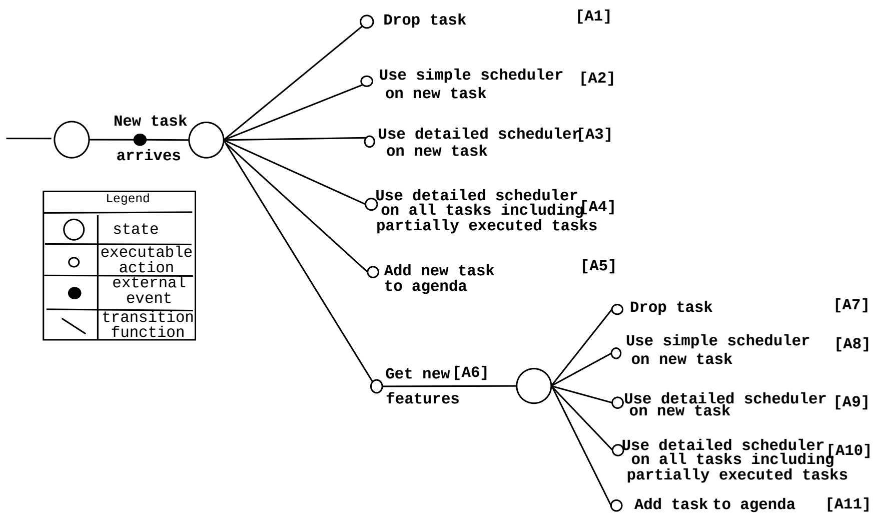

Drop task [A1]
Use simple scheduler on new task [A2]
Use detailed scheduler on new task [A3]
Use detailed scheduler on all tasks including partially executed tasks [A4]
Add new task to agenda [A5]
Get new features [A6]
Drop task [A7]
Use simple scheduler on new task [A8]
Use detailed scheduler on new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

Figure 3.2. Decision tree when a new task arrives

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 *Fred* 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 *Fred's* meta-level controller is invoked. *Fred* 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 *Fred* has gained a total of 6.0 utility points also. Based on its current state, *Fred's* meta-level control decision is to *Delay Explore Terrain task* and to add it to the Agenda List instead. *Fred* will continue execution of method *Focus Spectrometer on Rock*. When execution of this method is completed and if the NewTask List is empty, *Fred* 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. It is also inversely proportional to the autonomy of the agent, meaning, the agent chooses to give up more of its autonomy when it chooses higher effort alternatives. 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

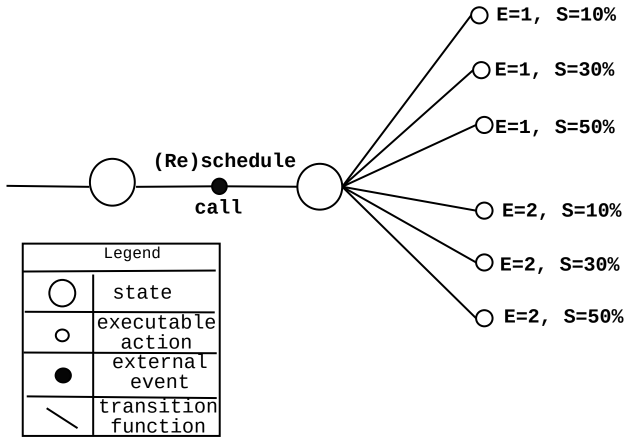


Figure 3.3. Decision tree for invoking the scheduler

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. This information is readily available to the sophisticated heuristic meta-level control strategy. The naive heuristic strategy uses a simple method of predicting arrival characteristics of future tasks based on past task arrival characteristics and is described in the next section. 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 amount of slack implicitly determines the autonomy of the agent. The greater the amount of slack in the schedule, the greater the level of autonomy since the agent has the flexibility of determining how to use its resources.

The decision tree describing the various action choices for this meta-level decision is shown in Figure 3.3. 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 *Fred* is in state *S2* and the time is 3. The new task *Explore Terrain* is to be scheduled using the detailed scheduler. *Fred'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
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 worth its while 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 3.4. This decision is made using the *MetaNeg* information described below.

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 [45]. The information gathering phase facilitates the negotiation phase and is modeled as a **MetaNeg** method in the task structure (see Figure 3.8) 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.

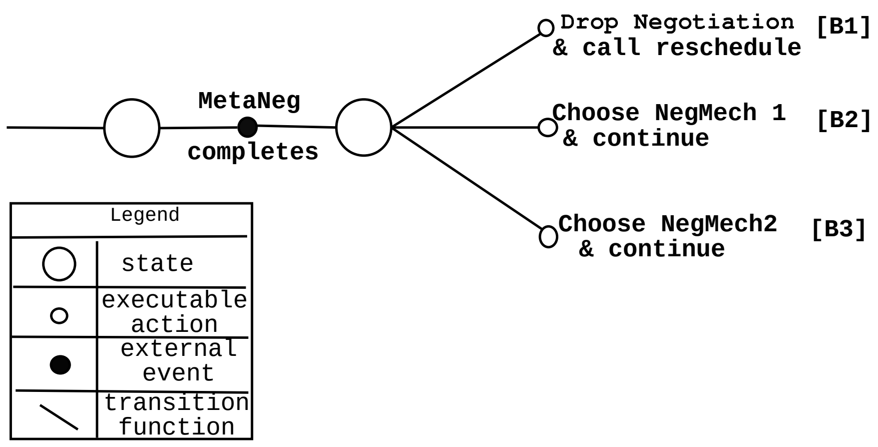

Drop Negotiation & call reschedule [B1]
MetaNeg completes
Choose NegMech 1 & continue [B2]
Choose NegMech2 & continue [B3]
Legend
state
executable action
external event
transition function

Figure 3.4. Decision tree on whether to negotiate and effort

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 completion 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.

When an agent decides to negotiate with another agent and abide by the results of the negotiation, it is effectively making a decision to relinquish some of its autonomy. The agent will have to reason about its tasks based not only on local constraints but also based on constraints that are outcomes of the negotiation action.

The following is an instance where *Fred* is in state *S10* at time 30. The new task *Explore Terrain* which arrived at time 15 with deadline 80 has been scheduled and the schedule is in the Execution List. *Fred's* meta-level controller is invoked. $\widehat{\text{F}}$red decides to begin execution of the Information Gathering Action (MetaNeg) in parallel with the *ExamineTerrain* action.

State S10:

CurrentTime :30
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{MetaNeg, NegMech2, ExamineTerrain, CollectSamples\}$

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 : 18.0
MLC Decision : **Begin execution of Information Gathering Action (MetaNeg) in parallel with ExamineTerrain**

The following is an instance where *Fred* is in state *S11* at time 33. The Information Gathering Action (MetaNeg) completes execution and *Fred's* meta-level controller is invoked. The following information is returned by the information gathering action: Agent *Barney* is executing high utility tasks, has deadlines which are far off and has a high amount of slack. Based on this information, *Fred* decides that it should negotiate with *Barney* using the NegMech2 protocol about the completion time of *Barney'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. When an agent decides to reschedule, it is also choosing to relinquish its autonomy and use of resources to re-process the task. Hansen's work [30] on meta-level control of anytime algorithms using a non-myopic stopping rule is described

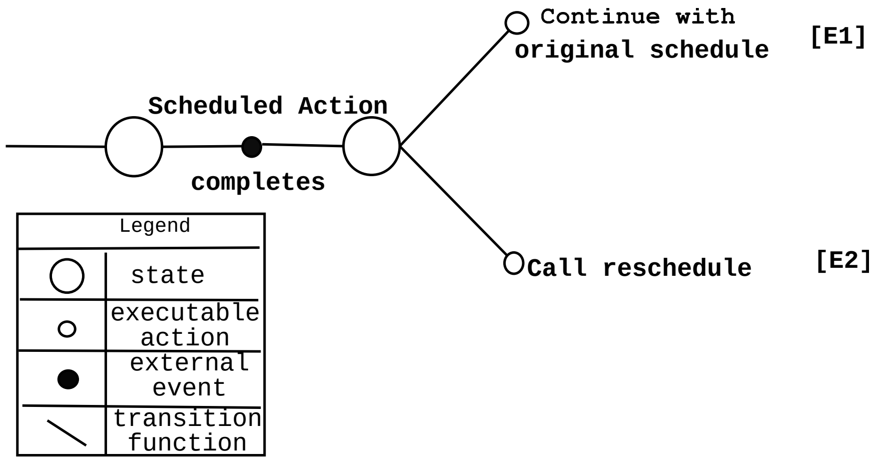


Figure 3.5. Decision tree when a domain action completes execution

in Chapter 2. 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. The decision to reschedule in this work 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 3.5.

The following is an instance where *Fred* 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. *Fred*'s meta-level controller is invoked to do a quick check to find out if the execution characteristics of this action is 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 FocusSpectrometeron-Rock**

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 3.6 describes the associated decision tree. When the agent decides to renegotiate, just like in the case of previous decision to negotiate, the agent is choosing to relinquish some of its autonomy to constraints determined by other agents.

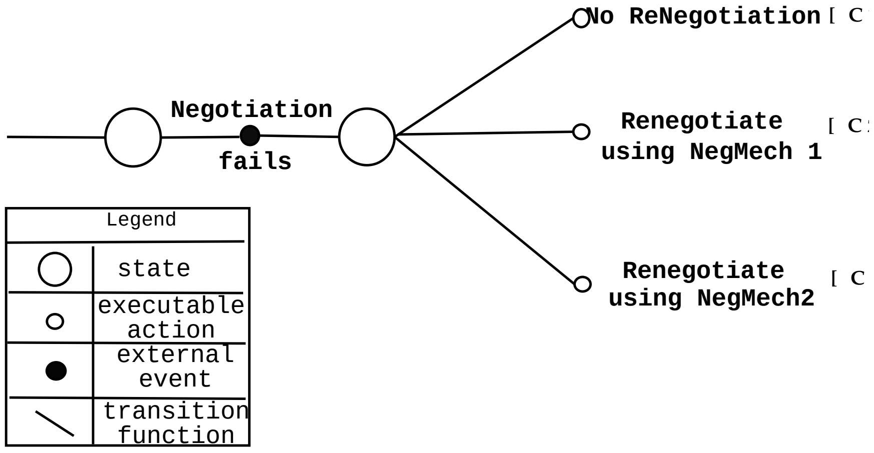


Figure 3.6. Decision tree on whether to renegotiate upon failure of previous negotiation

The following is an instance where *Fred* is in state *S15*, the time is 46 and method *NegMech2* has completed execution. *Fred's* meta-level controller is invoked. The results of the negotiation is that agent *Barney* will complete its method *Arrive at Location* at time 65 and this means *Fred'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.

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.

A detailed execution trace of *Fred's* behavior in a particular scenario is provided in Appendix B.

3.2 Agent State

The meta-level controller uses the current state of the agent to make appropriate decisions. In this dissertation, 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 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 twelve 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 $4^{12} = 2^{24}$, which is about a million states.

3.2.1 Abstract Representation of the State

The overhead of meta-level control activities is accounted for by the cost of state feature computation. The twelve 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 [2] characteristics and provide the meta-level controller with information 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

[2]Priority accounts for quality and deadline.

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 1.3, 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 3.1 will the actions in the MDP model. The probability transition function and the reward function will be determined by collecting data and estimating and learning them from the experience gained from the data.

3.2.2 Choosing State Features

The following are some characteristics of state features which allow for effective meta-level control in the task allocation domain described here.

1. Information on the status of tasks currently being processed and those which need future processing. Example: NewTaskList, AgendaList, ScheduleList, ExecutionList

2. Parameters of the objective function which should be maximized. Example: Utility of tasks

3. Parameters of the objective function which should be minimized. These include all the bounded resources of the environment. Example: Deadlines, Durations, Cost of tasks

4. Information on the environmental model if available. Example: Probability of arrival of tasks, task types and their deadline tightness

5. Parameters which affect the action choices. Example: Slack in the schedules

FeatureID	Feature	Complexity
F0	Status of Lists	Simple
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 3.1. Table of proposed state features, their description and category

6. Parameters which are computed based on real time performance. Example: Deviation from expected performance, Cost of decommiting from existing tasks

7. Information on other agents which influence local decisions. Example: Utility of tasks of other agents, Deadline Tightness of tasks belonging to other agents, Slack in schedules of other agents

Table 3.1 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.

F0: Status of Lists This is a simple feature which described the current status of all the lists the agent has to reason about. It is represented as a 4-tuple with the following entries: NewItemsList, Agenda, ScheduleList, ExecutionList, where each entry in the tuple contains the number of items on the corresponding list.
For example, $< 2, 0, 0, 1 >$ means there are two new items which have arrived from the environment and there is one task in execution.

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

3.2.3 Computation of State Features

The following is a formal description of how the low-level system parameters 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_{CT} is the total number of tasks that have arrived at the system from the start to current time CT. 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) $t^j, t^{j+1}, t^{j+2}...t^k$.*

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: $T1^A$ and $T1^B$ are two alternate plans to achieve task $T1$.
(25% 22 50% 50 25% 100) is the duration distribution of $T1^A$, which means that plan $T1^A$ 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 $T1^A$ has a utility distribution of (10% 30 90% 45). $T1^B$ has a duration distribution (50% 32 30% 40 20% 45) and utility distribution of (25% 20 75% 30).

$$UD_{T1^A} = (10\%\ 30\ 90\%\ 45)$$

$$DD_{T1^A} = (25\%\ 22\ 50\%\ 50\ 25\%\ 100)$$

$$UD_{T1^B} = (25\%\ 20\ 75\%\ 30)$$

$$DD_{T1^B} = (50\%\ 32\ 30\%\ 40\ 20\%\ 45)$$

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

Example : CT_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.

Definition 5: *R_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 R_m, the time required for completing the execution of the action m which is interrupted by a meta-level control event and CT_t, the time required for scheduling the new task.*

$$EST_t = AT_t + R_m + CT_t$$

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 $T1$ arrives at time 45 and has a deadline of 100. Also suppose the execution of method m is interrupted by the arrival of $T1$ and m still needs about 8 time units to complete execution. Suppose the time spent on scheduling $T1$ is 5 units. Then the maximum available duration for task $T1$ 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.*

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

Example: Suppose the maximum available duration for task $T1$ is 42. There is only one duration value in DD_{T1^A} which has a value less than 42 and that value is 22 and occurs 25% of the time.

$$PDL_{T1^A} = \frac{25}{100} = 0.25$$

There are two duration values in DD_{T1^B} which have a value less than 42 and they are 32 and 40 which occur 50% and 30% respectively in the distribution.

$$PDL_{T1^B} = \frac{50 \ + \ 30}{100} = 0.8$$

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.*

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

Example: For the above constraint where the maximum available duration for task $T1$ is 42

$$ED_{T1^A} = \frac{(\frac{25}{100} * 22\ )}{0.25} = 22$$

$$ED_{T1^B} = \frac{(\frac{50}{100} * 32\ +\ \frac{30}{100} * 40)}{0.8} = 35$$

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.*

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

Example: When the maximum available duration for task $T1$ is 42,

$$EU_{T1^A} = 0.25 * \frac{10}{100} * 30\ +\ 0.25 * \frac{90}{100} * 45 = 10.875$$

$$EU_{T1^B} = 0.8 * \frac{25}{100} * 20\ +\ 0.8 * \frac{75}{100} * 30 = 22$$

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 obtain the maximum utility in minimum duration within the given deadline.*

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

Example: Suppose the maximum available duration for task $T1$ is 42. Consider each of $T1$'s alternative plans which were described earlier. Plan $T1^A$'s expected utility to expected duration ratio is $\frac{10.875}{22}\ =\ 0.494$ and plan $T1^B$'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 is $T1^B$

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

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 $T1$ given a deadline of 100 and a maximum available duration of 42 is $\frac{22}{35} = 0.628$ which lies above the 66th percentile. $UD_{T1} = HIGH$

Definition 13: *There are some tasks which do not accrue utility uniformly, instead they get the total utility for the task only when execution of the task completes. Such tasks are called 0-1 utility tasks. The utility goodness of a 0-1 utility task in execution UD_t^{exe} of a 0-1 utility task t is the measure which determines how good the expected utility to remaining duration (R_t for task t at time CT) ratio of the task's preferred alternative is in relation to the expected utility to expected duration ratio of the preferred alternatives of all the other tasks which arrive at the system.*

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

Definition 14: *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* CML_t, during the execution of task t, is computed as follows:

$$CML_t = C_{ML} * \frac{N_{CT}}{CT} * MD_t$$

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. $CML_{T1} = 2 * \frac{4}{60} * 42 = 5.6$ 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.

$$MD_t^X = MD_t - CML_t$$

Example: From the previous example, the $MD_{T1}^X = 42 - 5.6 = 36.4$

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:

$$PDL^X_{T1^B} = 0.5, \quad ED^X_{T1^B} = 32, \quad EU^X_{T1^B} = 11.25, \quad UD^X_{T1} = \frac{EU^X_{T1^B}}{ED^X_{T1^B}} = 0.3451$$

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

$$UD^X_{T1} = LOW$$

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

Example: Since $(UD_{T1} = HIGH) \wedge (UD^X_{T1} = LOW)$, 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.

$$TD_{T1} = TIGHT$$

Definition 15: *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:

$$HPTS_A = \{T1\}$$

Definition 16: *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 17: *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.

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

Definition 18: *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 CML_t which was previously defined. The uncertainty in method durations is handled by the complex scheduler which reasons about uncertainty.

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

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.

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

Definition 19: *The expected utility of the current schedule scur (EU_{scur}) is provided by the domain scheduler when it completes constructing a schedule. The in-*

formation 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 [62], 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 CT 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

$$(\frac{EU_{scur}}{ED_{scur}} \gg \frac{EU_t}{ED_t}) \wedge (C_t >> SLACK_{scur}^{CT})$$

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 is probability of high priority tasks arriving during the scheduling duration is significantly high.

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

3.3 Detailed Execution Trace of Fred'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 Chapter 1 is provided. It provides a detailed view of the meta-level reasoning process of agent *Fred* 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 *Fred's* tasks are described in Figure 1.6. Suppose that along with information of the task structures, agent *Fred* also receives abstractions of tasks *AnalyzeRock* and *ExploreTerrain*. Abstractions for the tasks are described in the following table.

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

1. Should the method *CollectSamples* of task *ExploreTerrain* which is enabled by *Barney* be included in *Fred*'s schedule? This will determine the choice of the abstract alternative.

2. Should *Fred* 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

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 3.7. 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)

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 3.8. 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

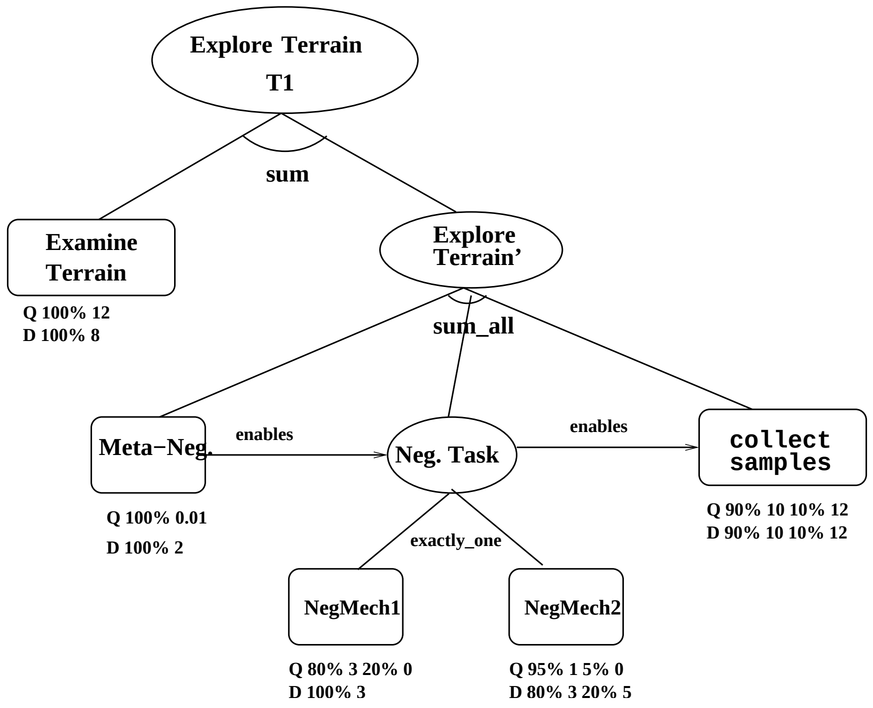


Figure 3.8. Task ExploreTerrain modified to include meta-negotiation action

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 completion of *NegTask* leads to the enablement of method X(*CollectSamples* in this example) as shown in the figure.

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

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

So task *AnalyzeRock* arrives at agent *Fred* 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 3.9 describes the *real* states agent *Fred* 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 Execution-List 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 (*Barney*) 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 *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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 learned 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: *Fred* 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**.

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 3.9. Agent *Fred's* state transitions

Time 13: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* is in state **S11** when execution of **MetaNeg** completes. *Fred*'s total utility is still 18. The information gathered by **MetaNeg** is as follows: *Barney* is executing schedule with HIGH expected utility/reward and the deadline tightness for these tasks is LOW. There is also high slack in *Barney's* schedule.
The following features are set:
F7: HIGH
Barney 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* 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: *Fred* is in state **S16** since method **ExamineTerrain** completes with utility 8. *Fred*'s total utility is 27. The features are
F8: LOW
The chosen action is **Begin execution of GettoRockLocation**.

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

Time 65: *Fred* 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: *Fred* 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: *Fred* 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.

3.4 Summary

This chapter 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. A formal method for determining agent state features is also described. The following two chapters 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 chapter or can be learned automatically as described in Chapter 5.

CHAPTER 4

HEURISTIC STRATEGIES

This chapter addresses the three following questions: Does meta-level control lead to better performance in rational agents situated in the task allocation and execution domain? Is it possible to construct a hand-generated meta-level control policy based on the high-level state features described in Chapter 3 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. Section 4.1 describes meta-level control within a single agent and will describe rules for three of the five meta-level questions. Experimental results describing the utility of meta-level control in a single agent are provided. Section 4.2 describes meta-level control within a multi-agent system consisting of two agents. The meta-level decision rules that facilitate coordination between the agents are presented. The experimental results describe the benefit of meta-level control for the entire multi-agent system.

4.1 Single Agent Heuristic Decision Making

4.1.1 Naive Heuristic Strategy

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

4.1.1.1 NHS Decision Rules for *Arrival of a new task* event

Suppose a new task arrives at time t. The agent has to decide whether to delay the reasoning about the task until later; never execute the task; execute the task immediately at arrival time by calling for a reschedule action or add the new task to the agenda. If more than one new task arrives at time t, the tasks will be evaluated individually and ordered based on the tightness of their deadlines.

As mentioned in Chapter 3, each of the event triggers has an associated decision tree which details the reasoning process. The reasoning process required to support each of the action choices in Figure 3.2 is described below.

1. Drop task and revert to original status [**A1**]: If utility of the new task is LOW and utility of current task set is HIGH or the deadline tightness of the task is TIGHT to even attempt to do the task, then drop the task with no reconsideration.

2. Do task now using simple scheduling i.e. abstraction-based alternative selection and call execution component[**A2**]: If utility of new task is HIGH, its deadline is TIGHT that detailed scheduling is not possible, the utility of current scheduled task set is LOW, then delay current schedule and use simple scheduling to process new task now.

3. Do task now using complex domain-level scheduler [**A3**]: If utility of new task is HIGH, its deadline is MEDIUM, yet far enough away to allow detailed scheduling of the single task, the utility of current scheduled task set is LOW, then delay current scheduled task set and use complex scheduling to schedule new task.

4. Call detailed scheduler on items on all lists[**A4**]: If both the utility of the new task and the agenda items are HIGH and comparable to the current schedule and the deadlines are MEDIUM, it is worthwhile to call the detailed scheduler on the new tasks, agenda and current schedule.

5. Add task to agenda[**A5**]: If the utility of the new task is HIGH, its deadline tightness is LOOSE, the utility of the current task set is also HIGH and there is enough time to schedule and successfully execute the new task after completion of the current task set, then delay reasoning of the new task for later by adding it to the agenda.

6. Get more features[**A6**]: If utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM or HIGH , and the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM or LOOSE and the slack in the schedule is MEDIUM or HIGH and the decommitment cost is MEDIUM or LOW, then obtaining more information on the slack distribution and/or commitment details will help the meta-controller to make accurate action choices.

    (a) Drop task and revert to original status[**A7**]: If the utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM, the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM and the decommitment cost is MEDIUM or LOW, and the slack in local schedule is LOW, then drop task.

(b) Do task now using simple scheduling [**A8**]: If the utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM or HIGH, the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM and the decommitment cost is MEDIUM or LOW, and the slack in local schedule is MEDIUM, then delay current schedule and use simple scheduling to do new task now.

(c) Accept task and call detailed scheduler on it [**A9**]: If the utility of of the new task and currently scheduled task set are both MEDIUM or HIGH, the deadline of new task and deadline of current schedule are both MEDIUM, the decommitment cost is MEDIUM and the slack in local schedule is HIGH, it is worthwhile to call the detailed scheduler on the agenda and current schedule.

(d) Call detailed scheduler on items on all lists[**A10**]: If the utility of the new task and the agenda items are both HIGH and the utility of the current schedule is also HIGH and the deadlines tightness is MEDIUM, the decommitment cost is MEDIUM and the slack in local schedule is HIGH, it is worthwhile to call the detailed scheduler on the new tasks, agenda and current schedule.

(e) Add task to agenda [**A11**]: If the utility of the new task is HIGH, its deadline is MEDIUM, the utility of the current task set is also HIGH and there is enough time to schedule and successfully execute the new task after completion of the current task set and slack in local schedule is LOW, then delay reasoning of the new task for later.

4.1.1.2 NHS Decision Rules for *Invoking domain level scheduler* event

Suppose there is a call to the scheduler either due to the arrival of a new task or due to a request to negotiate, the meta-level controller has to decide the appropriate parameters needed to call the scheduler. There are two parameters- **scheduler effort** which determines whether the new task or method can be easily fit in to the current schedule, fit with some modifications or requires a complete reschedule of the task set; and **slack** which determines whether to insert minimum slack in the schedule, making the schedule inflexible but very efficient or to insert a significant amount of slack and determine where it is appropriate that slack be introduced. A high amount of slack will trade-off efficiency for flexibility. Figure 3.3 describes the associated decision tree.

(a) Effort **E**: The *scheduling effort* parameter determines the amount of computational effort that should be invested by the scheduler. The parameter can be set to either *HIGH*, where a high number of alternative schedules are produced and examined if the task(s) to be scheduled have HIGH utility and LOOSE deadlines or *LOW*, where pruning occurs at a very early stage and hence few alternative schedules are compared, reducing the computational effort while compromising the accuracy of the schedule, if the task(s) to be scheduled have MEDIUM utility and MEDIUM deadline tightness.

(b) Slack **S**: If the tasks have high level of uncertainty in expected performance of local actions or in completion of commitments, and the deadlines are far enough in the future then the high slack option is chosen and detailed computation is done to decide where to appropriately fit the slack.

If the new task has HIGH uncertainty in its execution characteristics, set the slack to 50% of the total available time.

If the new task has MEDIUM uncertainty in its execution characteristics, set the slack to 30% of the total available time

In all other cases, set the slack to 10% of the total available time

4.1.1.3 NHS Decision Rules for *Significant deviation of online schedule performance* event

The meta-controller will monitor the actual performance characteristics of its method executions and determine when, if appropriate, it should call the scheduler to reevaluate the current status and determine an alternate course of action. Figure 3.5 describes the associated decision tree.

(a) Continue with original schedule [**E1**]: If the cumulative actual performance falls within a 10%[1] of the expected performance, in other words, if actual performance does not deviate too much from expected performance, then continue with the original schedule.

(b) Call reschedule [**E2**]: If the cumulative actual performance falls below the expected performance by 10%, then the rescheduler should be called.

4.1.2 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 dissertation, 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 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.

[1]this is a domain-dependent threshold value

4.1.2.1 SHS Decision Rules for *Arrival of a new task* event

The reasoning process required to support each of the action choices in Figure 3.2 using SHS is described below. As in the case of NHS decision rules, if a state feature is omitted in the rule, it means the feature can take any of its values.

1. Drop task and revert to original status [**A1**]:

   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 new task.*

   If utility of the new task is LOW and utility of current task set is HIGH or the deadline of task is TIGHT to even attempt to do the task, then drop the task with no reconsideration.

   If new task and current schedule have HIGH utility and MEDIUM or TIGHT deadlines; HIGH probability of high priority tasks arriving in the near future, then best action is *drop new task and continue with the current schedule*

2. Do task now using simple scheduling i.e. abstraction-based alternative selection and call execution component[**A2**]:

   If utility of new task is HIGH, its deadline is TIGHT such that detailed scheduling is not possible, the utility of current scheduled task set is significantly low, then delay current schedule and use simple scheduling to do new task now.

   If new task has LOW utility goodness and TIGHT deadline; LOW probability of high priority tasks arriving in the near future, then best action is *use simple scheduling to do new task now.*

   If a LOW utility new task is to be scheduled; LOW probability of a high priority task arriving in the near future, then best action is to use abstraction-based *simple scheduler* independent of the new task's deadline tightness.

3. Do task now using complex domain-level scheduler [**A3**]:

   If utility of new task is HIGH, its deadline is MEDIUM and far enough to allow detailed scheduling of the single task, the utility of current scheduled task set is MEDIUM, MEDIUM probability of high priority tasks arriving in the near future, then delay current scheduled task set and use complex scheduling to schedule new task.

   If new task has high priority; current schedule has LOW utility, TIGHT deadline; LOW probability of high priority tasks arriving in the near future, then best action is *drop current schedule and schedule the new task immediately.*

4. Call detailed scheduler on items on all lists[**A4**]:

   If the utility of the new task is HIGH and utility of the current schedule and agenda are either HIGH or MEDIUM and the deadlines are MEDIUM or LOOSE, it is worthwhile to call the detailed scheduler on the new tasks, agenda and current schedule.

5. Add task to agenda[**A5**]:

   If the utility of the new task is HIGH, its deadline is LOOSE, the utility of the current task set is also HIGH and its deadline is LOOSE or MEDIUM and probability of arrival of high priority tasks in the near future is MEDIUM or LOW then delay reasoning of the new task for later by adding it to the agenda.

   If new task has MEDIUM or LOW utility, MEDIUM or LOOSE deadline; current schedule has HIGH utility; LOW probability of high priority task arriving in the near future, then reasoning about the new task should be *delayed* till later.

6. Get more features[**A6**]:

   If utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM or HIGH , and the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM or LOOSE and the slack in the schedule is MEDIUM or HIGH and the decommitment cost is MEDIUM or LOW, and the probability of arrival of high priority tasks in the near future is LOW, then obtaining more information on the slack distribution and/or commitment details will help the meta-controller to make accurate action choices.

   (a) Drop task and revert to original status[**A7**]: If the utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM, the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM and the decommitment cost is MEDIUM or LOW, and the slack in local schedule is LOW, then drop task.

   (b) Do task now using simple scheduling [**A8**]: If the utility of tasks in the agenda and utility of currently scheduled task set are both MEDIUM or HIGH, the deadline of tasks in the agenda and deadline of current schedule are both MEDIUM and the decommitment cost is MEDIUM or LOW, and the slack in local schedule is MEDIUM, then delay current schedule and use simple scheduling to do new task now.

   (c) Accept task and call detailed scheduler on it [**A9**]: If the utility of of the new task and currently scheduled task set are both MEDIUM or HIGH, the deadline of new task and deadline of current schedule are both MEDIUM, the decommitment cost is MEDIUM and the slack in local schedule is HIGH, it is worthwhile to call the detailed scheduler on the agenda and current schedule.

   (d) Call detailed scheduler on items on all lists[**A10**]: If the utility of the new task and the agenda items are both HIGH and the utility of the current schedule is also HIGH and the deadlines tightness is MEDIUM, the decommitment cost is MEDIUM and the slack in local schedule is HIGH, it is worthwhile to call the detailed scheduler on the new tasks, agenda and current schedule.

(e) Add task to agenda [**A11**]: If the utility of the new task is HIGH, its deadline is MEDIUM, the utility of the current task set is also HIGH and there is enough time to schedule and successfully execute the new task after completion of the current task set and slack in local schedule is LOW, then delay reasoning of the new task for later.

4.1.2.2 SHS Decision Rules for *Invoking domain level scheduler* event

Figure 3.3 describes the associated decision tree for determining the scheduling effort and slack parameters

(a) Effort **E**: The *scheduling effort* parameter determines the amount of computational effort that should be invested by the scheduler. The parameter can be set to either *HIGH* or *LOW*.

If the utility of the new task is MEDIUM or LOW, and its deadline tightness is MEDIUM, the utility of the current task is MEDIUM and the deadline tightness is MEDIUM or LOW and there is a LOW probability of a high priority task arriving in the near future, then the scheduler effort should be set to LOW.

In all other cases, the scheduler effort is set to high.

(b) Slack **S**: Slack allows for flexibility in the schedule to handle unexpected events. It can be set to 10%, 30% and 50% of the total available time.

If the new task has HIGH uncertainty in its execution characteristics and there is a MEDIUM to high probability of arrival of high priority tasks in the near future, set the slack to 50% of the total available time.

If the new task has MEDIUM uncertainty in its execution characteristics and there is a MEDIUM to high probability of arrival of high priority tasks in the near future, set the slack to 30% of the total available time

In all other cases, set the slack to 10% of the total available time

4.1.2.3 SHS Decision Rules for *Significant deviation of online schedule performance* event

Figure 3.5 describes the associated decision tree.

(a) Continue with original schedule [**E1**]:

If the cumulative actual performance falls within a 10% (a domain-dependent value) of the expected performance, in other words, if actual performance does not deviate too much from expected performance, then continue with the original schedule.

If the cumulative actual performance falls below the expected performance by 10%, and there is HIGH probability of arrival of high priority tasks in the near future then continue with the original schedule..

(b) Call reschedule **[E2]**:

If the cumulative actual performance falls below the expected performance by 10% and there is LOW to MEDIUM probability of arrival of high priority tasks in the near future, then the rescheduler should be called.

4.1.3 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. The deterministic strategy uses a fixed choice of meta-level action. When a new task arrives, this strategy always chooses to perform complex scheduling on the new task along with the tasks in the current schedule and tasks in the agenda. The scheduler is invoked with a fixed effort level of high and fixed slack amount of 10% of the total schedule duration. 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 described in this dissertation, 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 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 agents in the experimental test-bed were implemented using the Java Agent Framework (JAF) framework and situated in the Multi-Agent Survivability Simulator (MASS) environment. A detailed description of JAF and MASS is provided in Appendix A. Each agent simulation was run on a Intel Pentium(R) machines with four 1.80GHz 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 alternatives is similar to the AnalyzeRock task (Figure 1.6)described in

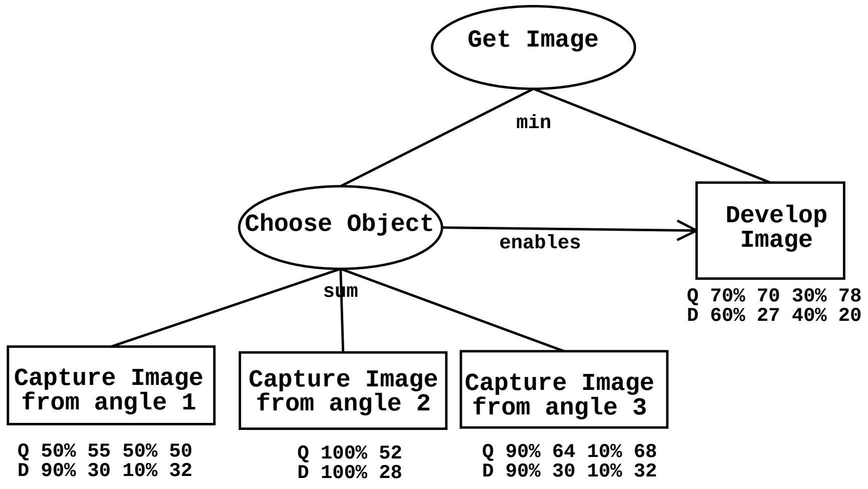


Figure 4.1. A Complex Task in a Single Agent Environment

Chapter 1. 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 4.1. 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 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.

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 4.1. 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)

The experimental results described in Table 4.1 show the performance of the various strategies in an 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 an average duration of 22 time units and complex tasks have an average duration of 32 time units. 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

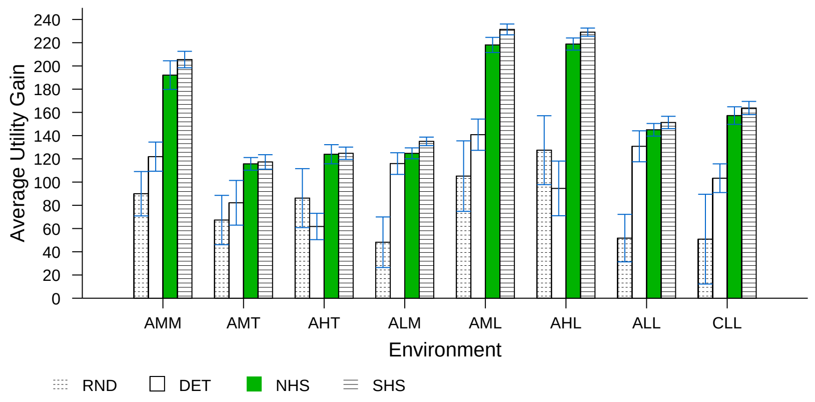


Figure 4.2. Average Utility Comparison between Heuristic Strategies and Baseline Strategies over 8 different environments. The error bars are one standard deviation above and below each mean

actions instead of being spent the utility deriving domain actions. Row 3 shows that the duration spent on control actions by NHS is significantly ($p < 0.05$) higher than that of SHS. The deterministic strategy does not ever reschedule but invests a lot of time on control actions since the fixed strategy is time-intensive. The random strategy spends the least amount of time on control (11.77%) because it attempts relatively few tasks (there is a high probability of a task being dropped randomly upon arrival).

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 ask 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 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 4.2 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 4.2 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

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 4.2. Utility Comparisons over a number of environments

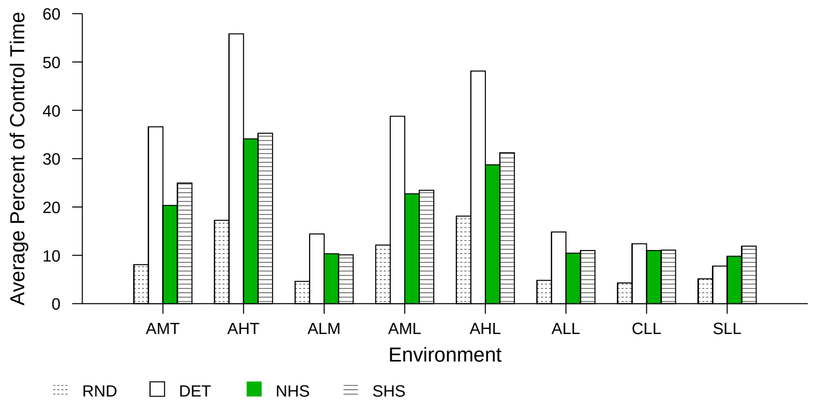


Figure 4.3. Average Percent Control Time Comparison between Heuristic Strategies and Baseline Strategies over 8 different environments

respect to the deterministic algorithm. Column 8 named p3 shows the statistical significance of NHS with respect to the deterministic algorithm.

The reason for the improved performance by the heuristic strategies when compared to the deterministic and random strategies is found in Figure 4.3 which shows the percent of control time comparisons over the same set of environments. As described in Chapter 3, 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

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

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 4.3 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 4.2 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.

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 4.4. 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 4.4 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.

4.2 Multi-Agent Heuristic Decision Making

4.2.1 Naive Heuristic Strategy

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 task completion. The following are the heuristics for the two additional meta-level decisions.

4.2.1.1 NHS Decision Rules for *Presence of task requiring Negotiation* event

Figure 3.4) describes the action set for this event. Suppose there is a subtask or method in the currently scheduled task 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 worth its while to even initiate negotiation and if so, what kind of negotiation mechanism to use. The information gathering action, *MetaNeg*, when executed, will gather information on the state of the non-local agent that is being considered for negotiation. The meta-meta decision on whether to execute *MetaNeg* is made by the domain-level scheduler which will compare the alternate ways of achieving the task and to determine whether the approach to solve parts of the task through negotiation seems advantageous over other approaches.

1. No negotiation and call reschedule if needed[**B1**]: If utility of the task set of the non-local agent is HIGH, task utility of the new task is LOW, and there is not enough slack to fit the new task in the non-local schedule, then don't negotiate.

2. Negotiate with NegMech1 [**B2**] : If utility of the task set of the other agent is MEDIUM, flexibility of the schedule of the non-local agent is MEDIUM and utility of the new task is HIGH, then the agent should negotiate with the single-shot negotiation mechanism NegMech1.

3. Negotiate with NegMech2 [**B3**] : If utility of the task set of the other agent is LOW, flexibility of the schedule of the non-local agent is HIGH and utility of the new task is HIGH, then the agent should negotiate using NegMech2 which is a multi-try negotiation mechanism.

4.2.1.2 NHS Decision Rules for *Failure of Negotiation* event

Suppose there is a subtask or method in the currently scheduled task 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 mechanism should it use? Figure 3.6 describes the associated decision tree.

1. No Renegotiation and reschedule is needed[**C1**]: If there is no time left for renegotiation then drop negotiation and the related method from the schedule.

2. Renegotiate using NegMech1 [**C2**]: If utility of the task set of the other agent is MEDIUM, flexibility of the schedule of the non-local agent is MEDIUM and utility of the new task is HIGH, then the agent should negotiate with NegMech1.

3. Renegotiate using NegMech2 [**C3**]: If utility of the task set of the other agent is LOW, flexibility of the schedule of the non-local agent is HIGH and utility of the new task is HIGH, then the agent should negotiate using NegMech2.

4.2.2 Sophisticated Heuristic Strategy

The following are the heuristics for the two additional meta-level decisions necessary in multi-agent contexts.

4.2.2.1 SHS Decision Rules for *Presence of task requiring Negotiation* event

Figure 3.4) describes the action set for this event. The event occurs when the *MetaNeg* information gathering action completes execution.

1. No negotiation and call reschedule if needed[**B1**]:

   If utility of the task set of the non-local agent is HIGH, task utility of the new task is LOW, if slack in the non-local schedule is LOW, and there is MEDIUM to HIGH probability of arrival of high priority tasks in the near future, then don't negotiate.

2. Negotiate with NegMech1 [**B2**] :

   If utility of the task set of the other agent is LOW, slack in the schedule of the non-local agent is HIGH, utility of the new task is HIGH and LOW probability of arrival of high priority tasks in the near future, then the agent should negotiate using NegMech1.

   If utility of the task set of the other agent is LOW or MEDIUM, flexibility of the schedule of the non-local agent is HIGH, utility of the new task is MEDIUM, and LOW probability of arrival of high priority tasks in the near future, then the agent should negotiate using NegMech1.

3. Negotiate with NegMech2 [**B3**] :

   If utility of the task set of the other agent is LOW, slack in the schedule of the non-local agent is HIGH and utility of the new task is HIGH, its deadline tightness is MEDIUM or FAR, and LOW or MEDIUM probability of arrival of high priority tasks in the near future, then the agent should negotiate using NegMech2.

4.2.2.2 SHS Decision Rules for *Failure of Negotiation* event

Figure 3.6 describes the decision tree for what to do when a previous negotiation fails.

1. No Renegotiation and reschedule is needed[**C1**]:

   If there is no time left for renegotiation then drop negotiation and the related method from the schedule.

2. Renegotiate using NegMech1 [**C2**]:

   If utility of the task set of the other agent is LOW or MEDIUM, flexibility of the schedule of the non-local agent is HIGH, utility of the new task is MEDIUM, and LOW probability of arrival of high priority tasks in the near future, then the agent should renegotiate using NegMech1.

3. Renegotiate using NegMech2 [**C3**]:

   If utility of the task set of the other agent is LOW, slack in the schedule of the non-local agent is HIGH and utility of the new task is HIGH, its deadline tightness is MEDIUM or FAR, and LOW or MEDIUM probability of arrival of high priority tasks in the near future, then the agent should renegotiate using NegMech2.

4.2.3 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 deterministic strategy in the multi-agent setup uses a fixed choice of meta-level action just as in the single agent set-up. 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. When negotiation is required for a task, the agent always chooses to negotiate with another agent and if negotiation fails, the agent does not choose to negotiate again. The random strategy randomly chooses its actions for each of the five meta-level control decisions.

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 described in this dissertation, 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.

The task environment generator in the multi-agent setup also 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 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 1.6) described in Chapter 1. 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 task described in Figure 1.6 and the other has structure similar to that of GetImage task described in Figure 4.1. 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 4.4 and Table 4.5. Performance comparison of the various strategies in an environment, AMM, over a number of dimensions are provided. The

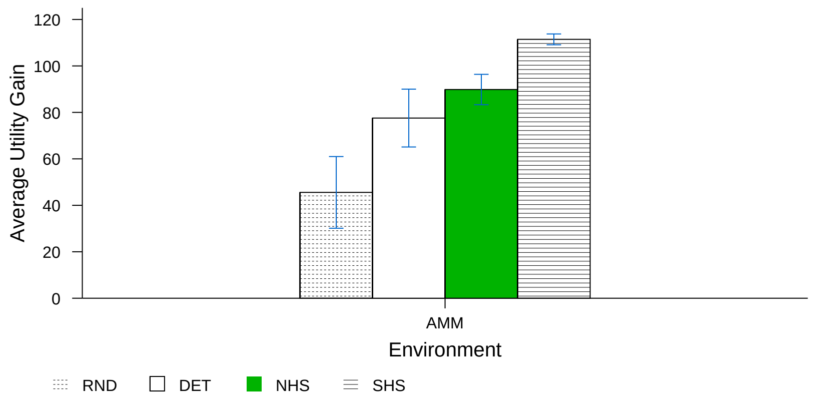


Figure 4.4. Average Utility Comparison between Heuristic Strategies and Baseline Strategies in a Multiagent environment. The error bars are one standard deviation above and below each mean

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 dissertation. Further experimental studies to establish the advantage of meta-level control in multi-agent systems are ongoing.

4.3 Summary

This chapter presents 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 described in the previous chapter are good indicators of the agent state and facilitate effective meta-level control; the heuristic strategies are good indicators of 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.

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 4.5. 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)

CHAPTER 5

REINFORCEMENT LEARNING STRATEGY

Can an agent automatically learn meta-level control policies for specific environments based on the high-level state information described in Chapter 3? Does this learned policy outperform the corresponding hand-crafted policy for that environment as described in Chapter 4? These are the two questions addressed in this chapter.

The high-level goal of this work 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 5.3 describes the construction of a Markov decision process-based 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 5.4 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.

5.1 Reinforcement Learning

Reinforcement Learning [1, 36, 76, 77, 75, 84, 86] 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.

5.2 Markov Decision Processes

The following is a formal specification which captures the critical aspects of the problem facing a learning agent interacting with the environment to achieve its goals.

Markov decision processes (MDPs) [56] are the standard reinforcement learning framework. An MDP is defined via its state set S, action set A, transition probability matrices P, and reward matrices R. On executing action a in state s the probability of transitioning to state s' is denoted $P^a(ss')$ and the expected reward associated with that transition is denoted $R^a(ss')$. The rewards are non-negative for all transitions in this work.

A rule for choosing actions is called a *policy*. Formally it is a mapping π from the set of states S to the set of actions A. If an agent follows a fixed policy, then over many trials, it will receive an average total reward which is known as the *value* of the policy. In addition to computing the value of a policy averaged over all trials, we can also compute the value of a policy when it is executed starting in a particular states s. This is denoted $V^\pi(s)$ and it is the expected cumulative reward of executing policy π starting in state s. This can be written as

$$V^\pi(s) = E[r_{t+1} + r_{t+2} ... | s_t = s, \pi]$$

where r_t is the reward received at time t, s_t is the state at time t, and the expectation is taken over the stochastic results of the agent's actions.

For any MDP, there exist one or more optimal policies which we will denote by $\pi*$ that maximize the expected value of the policy. All of these policies share the same optimal value function, which is written V^* The optimal value function satisfies the Bellman equations [3]:

$$V^*(s) = \max_a \Sigma_{s'} P(s'|s,a)[R(s'|s,a) + V^*(s')]$$

where $V^*(s')$ is the value of the resulting state s'. The sum on the right-hand-side is the expected value of the one step reward $R(s'|s,a)$ plus the value of the nest state s', which is the same as the backed-up value of a one-step lookahead search, and the max_a is choosing the action with the best backed-up value. The sum is named $Q^*(s,a)$

$$Q^*(s,a) = (\Sigma_{s'} P(s'|s,a)[R(s'|s,a) + V^*(s')]$$

This is the expected total reward that will be received when the agent performs action a in state s and then behaves optimally thereafter. By substituting this into the Bellman equation, it is shown that the value function is just the maximum (over all the actions) of the Q function.

$$V^*(s) = \max_a Q^*(s,a)$$

Consequently, this can be substituted in to the Q equation to obtain the Q version of the Bellman equation.

$$Q^*(s,a) = \Sigma_{s'} P(s'|s,a)[R(s'|s,a) + \max_a Q^*(s',a')$$

5.3 Single Agent Meta-Level Control

The learning approach adopted for the meta-level control problem is based on the algorithm developed in [72] 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 Chapter 4, each episode lasting 500 simulation time clicks takes about 180 seconds on a Intel Pentium(R) machine with four 1.80GHz processors running linux. It takes about 150 hours to obtain data from 3000 training episodes making data collection quite expensive.

As described in previous chapters, the 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 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 [72], 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 [75]. 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 [75]:

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

When the value iteration algorithm converges[1], 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

5.3.1 Experiments

The experimental setup is as described in the previous chapter. 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 described in this dissertation, 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 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

[1]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

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 agents in the experimental test-bed were implemented using the Java Agent Framework (JAF) framework and situated in the Multi-Agent Survivability Simulator (MASS) environment. A detailed description of JAF and MASS is provided in Appendix A. Each agent simulation was run on a Intel Pentium(R) machines with four 1.80GHz 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 alternatives is similar to the AnalyzeRock task (Figure 1.6)described in Chapter 1. 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 4.1. 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 tightness is set to high, the maximum available duration is between 80% and 100% of the expected duration of the task.

The training data for the RL strategy consisted of 3000 episodes. After the 3000 episodes were completed, the estimated transition probabilities and reward function

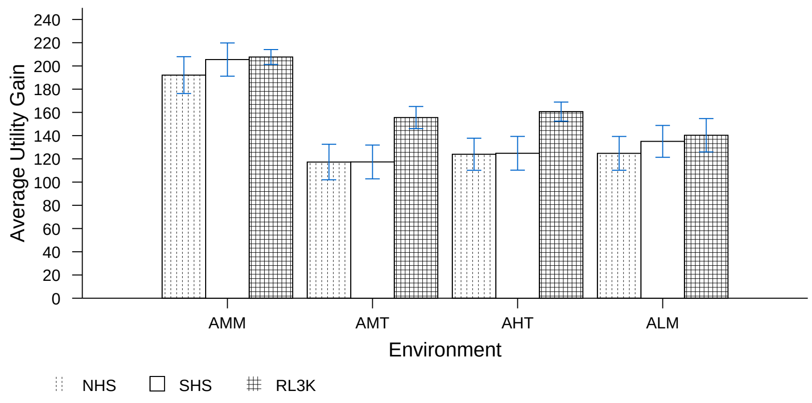


Figure 5.1. Utility Comparison of Learning Method to Heuristic Strategies for four different environments. The error bars are one standard deviation above and below each mean

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.

The results described in Figure 5.1 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). In loosely constrained environments, agents have enough resources to complete tasks successfully within the deadlines with out too much contention of resources. In all four environments, the RL strategy using the policy based on 3000 training episodes did as well as if not significantly better ($p < 0.05$) than the SHS with respect to utility but had significantly lower control duration.

Figure 5.2 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.

Learning Curve Saturation: Figure 5.3 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 necessar-

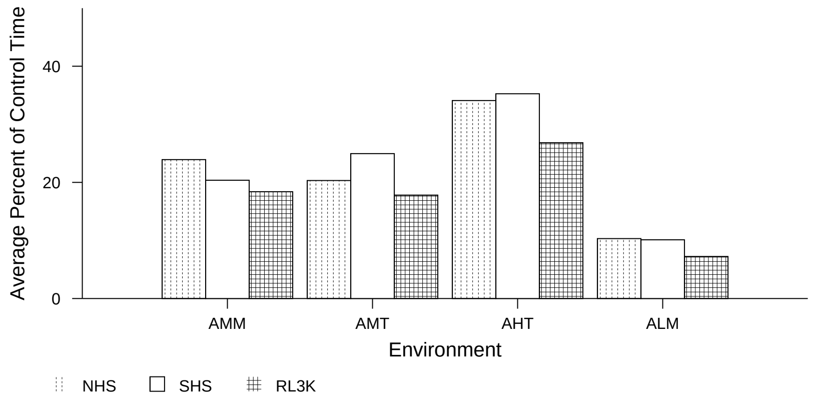


Figure 5.2. Control Time Comparison of Learning Method to Heuristic Strategies for four different environments

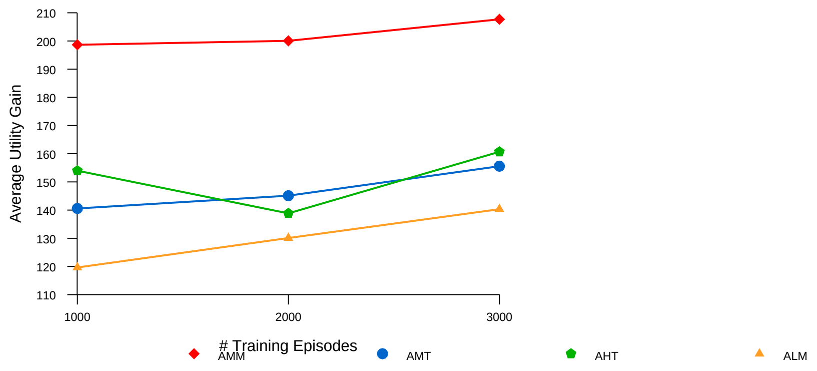


Figure 5.3. Relation of Average Utility to Increasing Training Data

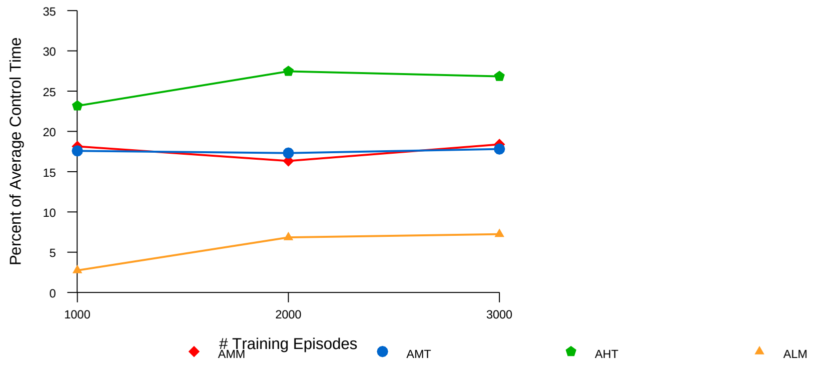


Figure 5.4. Relation of Control Time Durations to Increasing Training Data

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 5.1. 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;

ily 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 work. 3000 seemed to be a good threshold for training size for the four environments described. The dip the curve for AHT at episode 2000 is a local minima which occurred because of the tightly constrained environment. Figure 5.4 describes the relation of the percent of control durations to increasing training size.

Table 5.1 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

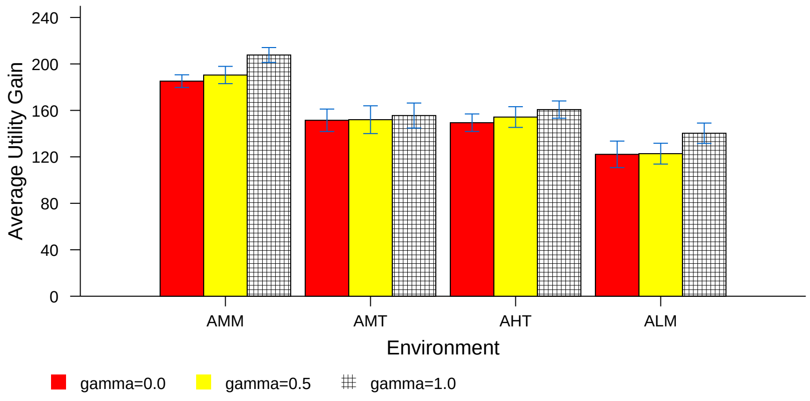


Figure 5.5. Utility Gains with varying discount rate (γ = 0.0, 0.5, 1.0). The error bars are one standard deviation above and below each mean

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 5.5 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 300 test episodes and the average utilities over these 300 episodes are computed. Table 5.2 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.

5.4 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. In this dissertation, the interaction between 2 agents *Fred* and *Barney* is studied. The multi-agent aspect of the problem arises only when there is task requiring coordination

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 5.2. Comparison of utility gain and percent of control time for four different environments while varying the discount rate (γ = 0.0, 0.5, 1.0)

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 [49]. The experiments in this chapter describe results on the convergence rates of the policies of the two agents in simple scenarios.

5.4.1 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 described in this dissertation, 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.

The task environment generator in the multi-agent setup also 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 $dl \epsilon \{tight(T), medium(M), loose(L)\}$.

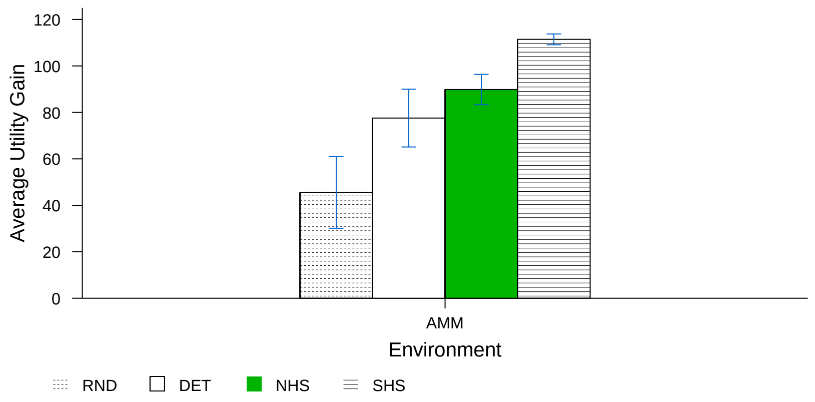


Figure 5.6. Average Utility Comparison between Heuristic Strategies and RL Strategy (300 training episodes) in a Multiagent environment. The error bars are one standard deviation above and below each mean

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 1.6) described in Chapter 1. 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 task described in Figure 1.6 and the other has structure similar to that of GetImage task described in Figure 4.1. 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 5.6 and Table 5.3. Agent *Fred's* was fixed to the best policy it was able to learn in the single agent environment. Agent *Barney* 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 dissertation. Further experimental studies to establish the advantage of automatically learning meta-level control policies in multi-agent systems are ongoing.

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

Table 5.3. 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;

5.5 Summary

This chapter describes 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 [72] 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.

CHAPTER 6

CONCLUSIONS

This dissertation explores the issue of meta-level control in complex agents situated in social and dynamic environments. As discussed in Chapter 1, 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 dissertation 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 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.*

6.1 Main Results

A meta-level agent architecture for bounded-rational agents which supports alternative approaches for deliberative computation is described in Chapter 3. 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. This chapter also included a discussion of how adjustable autonomy emerges as a feature of an agent equipped with meta-level reasoning. 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.

Chapter 4 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. I 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.

Chapter 5 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 [72] 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 improvement in performance was about 60% in the best case over the heuristic strategies. The sequential effects of the problem domain was verified by showing that varying the value of future rewards significantly affects the agent's performance.

6.2 Applying this work

This dissertation 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 dissertation 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 described in this dissertation allows for meta-level control in uncertain environments whose model is not available. The empirical reinforcement learning algorithm described in this dissertation, allows the agent to construct a partial model of the environment and use the information to define effective action policies.

Additionally, the dissertation 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.

6.3 Future Extensions

1. Feature Learning

   In Chapter 4, I present the general criteria which can be used to choose high-level state features which facilitate meta-level control. I'm interested in equipping the agents to learn these state features automatically. This would allow agents to fine-tune the meta-level control process to particular application domains and also determine the subset of features that are critical to specific meta-level decisions. Standard learning techniques like C4.5 could be used for the feature learning process.

   In this work, each state feature can take on one of three qualitative values. The number of values was manually set although the mapping from quantitative values to these qualitative buckets was determined offline for each environment. It will also be interesting to allow the agent to learn the optimal number of qualitative values as well as the mapping from qualitative to quantitative values for each environment.

2. Scalability, Robustness and Safety

   Uncertainty is ubiquitous in the problem domains discussed in this dissertation. It is necessary to leverage this uncertainty at all levels of reasoning [57]. It is also important that meta-level control take into consideration the robustness and safety of its systems while making its decisions. Some domains are more risk-averse than others and the learning mechanism in the meta-level control can be provided the knowledge of scenarios which will be unacceptable in both the training and testing phase. The meta-level control strategies described here are scalable in theory. I plan to verify this by scaling up the episode lengths; increasing the number of agents in the multi-agent system; and increasing the durations of tasks so that there are more interactions and resource contentions among tasks.

3. Abstraction in Control Reasoning:

   Abstraction information of tasks allows meta-level control to make quick and effective decisions on tasks. Abstraction information such as expected quality distribution, expected duration distribution, and expected duration uncertainty, of each task are determined manually in this work. I would like to construct algorithms which can automate this task abstraction process. There are two types of analysis that seem feasible: semantic analysis of tasks where the clustering algorithms can be used to cluster parts of the tasks structures which do not need explicit distinction and those which do; and statistical analysis where the agent will systematically vary the performance criteria and gather statistics on the task performance characteristics and then construct task abstractions. It will be interesting to study if the two types of analysis lead to equivalent task abstractions.

4. Meta-level Control in Multi-Agent Organizational Design

   One of the most important characteristics of large-scale multi-agent systems is their organization - the description of how, when and with whom member agents should interact. The type, number and qualities of these relationships can have a large impact on the effectiveness of both individual agents and the system as a whole. In dynamic or failure-prone environments, the task of keeping the organization efficient and coherent can be quite difficult. The cost of reorganization can be significant in some cases. Meta-level control techniques which reason about costs and resource consumption like those described in this dissertation can make quick, inexpensive decisions on when it is necessary to reorganize and how much effort to invest into the reorganization and which parts of the organization need to be re-evaluated.

5. More Complex Alternatives for Control

   I would also like to construct a wider range of alternatives for scheduling/planning as well for interactions which require coordination with other agents. The ability of meta-level control and state features to scale up to more complex environments will be studied.

6. Performance Upper Bounds

   Computing an optimal meta-level control policy for the problem described in this dissertation is NP-hard. A very loose upper bound of utility gained would the sum of the expected utilities of all the tasks which arrive at the agents. However, the deadlines and the finite horizon of the problem do not allow for all tasks to be completed even in the most optimistic case. It is important to compare the performance of the learning and heuristic strategies to an approximately optimal policy. A branch-and-bound technique to determine a more feasible upper bound for agent performance is currently being designed.

APPENDIX A

TOOLS AND FRAMEWORKS

TÆMS Modeling Language

TÆMS (Task Analysis, Environment Modeling, and Simulation) is a domain independent task modeling framework used to describe and reason about complex problem solving processes. TÆMS models are used in multi-agent coordination research [18, 83] and are being used in many other research projects, including: Cooperative-Information-Gathering [53, 43], collaborative distributed design [20], distributed situation assessment [11], surviveable systems [81], multi-agent diagnoses [2], intelligent environments [44], hospital patient scheduling [17], and coordination of software process [35]. Typically a problem solver represents domain problem solving actions in TÆMS, possibly at some level of abstraction, and then passes the TÆMS models on to agent control problem solvers like the multi-agent coordination modules or the Design-to-Criteria scheduler.[1]

TÆMS models are hierarchical abstractions of problem solving processes that describe alternative ways of accomplishing a desired goal; they represent major tasks and major decision points, interactions between tasks, and resource constraints but they do not describe the intimate details of each primitive action. 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. Duration describes the amount of time that the action modeled by the method will take to execute and cost describes the financial or opportunity cost inherent in performing the action. With the addition of uncertainty modeling, the statistical characteristics of the three dimensions are described via discrete probability distributions associated with each method. The uncertainty representation is also applied to task interactions like enablement, facilitation and hindering effects.[2] Thus agents may not only reason about the certainty of actions, e.g., "method A will fail 10% of

[1]In the process work, a translator transforms and abstracts process programs into TÆMS task structures for scheduling and coordination.

[2]Facilitation and hindering task interactions model soft relationships in which a result produced by some task may be beneficial or harmful to another task. In the case of facilitation, the existence of the result, and the activation of the nle generally increases the quality of the recipient task or reduces its cost or duration.

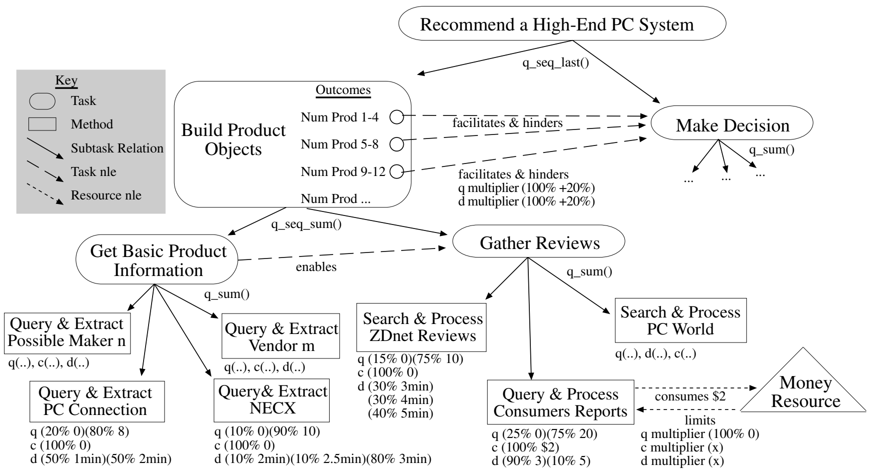


Figure A.1. Information Gathering Task Structure

the time," but also with respect to the interactions, e.g., "10% of the time facilitation will increase the quality by 5% and 90% of the time it will increase the quality by 8%," and the joint of these two. (Since interaction effects are dependent on the quality of the originator of the effect.) The quantification of methods and interactions in TÆMS is not regarded as a perfect science. Task structure programmers or problem solver generators *estimate* the performance characteristics of primitive actions. These estimates can be refined over time through learning and reasoners typically replan and reschedule when unexpected events occur.

To ground further discussion, consider Figure A.1, which is a slightly more complete view of the information gathering task structure introduced in Figure A.2. The top-level task in this structure is *Recommend-a-High-End-PC-System* and it has two subtasks: one that pertains to finding information about products and constructing models of them, *Build-Product-Objects*, and one for making the decision about which product to purchase, *Make-Decision*. The two tasks are governed by a *seq_last()* qaf. Qafs specify how the quality of the subtasks is related at the parent task. They may also specify orderings among the subtasks. Let T denote a task, c_i denote one of its children, and let n denote the number of children of T. Let q denote the quality of the item in question, e.g., T_q is the quality of the task and c_{i_q} is the quality of the *ith* child of T. In TÆMS, the quality of any task or method before performance (or after failure) is zero. A sampling of the qafs defined in TÆMS includes:

- *sum*: $T_q = \sum_{i=1}^{n} c_{i_q}$ and any of the subtasks may be performed (power-set minus empty-set) in any order.
- *sum_all*: $T_q = \sum_{i=1}^{n} c_{i_q}$ and all subtasks are to be performed in any order.
- *min*: $T_q = min(c_{0_q}, c_{1_q}, .., c_{n_q})$ and all subtasks are to be performed in any order. Since all tasks have zero initial quality, failure to perform a given child under a *min* results in zero quality for the parent task.

- *max*: $T_q = max(c_{0_q}, c_{1_q}, .., c_{n_q})$ and any number of subtasks may be performed in any order, though generally only one task is selected.
- *exactly_one*: $T_q = (c_{0_q}\ EXOR\ c_{1_q}\ EXOR\ c_{n_q})$ and only one of the subtasks may be performed.
- *seq*: $T_q = c_{n_q}$ and all subtasks must be performed in order.
- *seq_sum, seq_min, seq_max*: The *seq* prefix in this case denotes sequence preference and that all subtasks must be performed; the suffix denotes the function to perform with the resultant qualities, e.g., *seq_sum* indicates $T_q = \sum_{i=1}^{n} c_{i_q}$

Recommend-a-High-End-PC is thus performed by performing each of its subtasks, in order, and its quality is determined by the *Make-Decision* subtask. This models the fact that the decision process takes into consideration the quality, coverage, and certainty of the information used to make the decision and reflects these attributes in the quality of its output. As discussed, *Build-Product-Objects* is performed by performing each of its child tasks, in order, and its quality is the sum of its children's qualities. In contrast, *Get-Basic* and *Gather-Reviews* can be achieved by performing any one or more of their respective child tasks. Note the *enables* interaction between *Get-Basic* and *Gather-Reviews*. This nle models a hard precedence relationship between the tasks – the agent must first successfully learn about products before it can locate reviews for them. In TÆMS, task interactions are triggered by conditions in the originator and the effects of the interactions are reflected in the quality, cost, and duration distributions of the recipient. With the addition of uncertainty to TÆMS, soft interaction effects like facilitation and hindering, are also quantified via probability distributions. Task interactions in TÆMS include: *facilitates*, *hinders*, *bounded facilitates*, *sigmoid*, *enables*, and *disables*.

Outcomes, in TÆMS model situations in which a given method has different classes of possible results, each class having its own distinct quality, cost, and duration characteristics and possibly even its own interactions with other tasks. The *Build-Product-Objects* task in Figure A.1 illustrates the outcomes construct[3]; the outcomes serve to indicate the number of objects generated during the information gathering phase. Attached to each of these outcomes are hindering and facilitation soft nles that affect the quality, cost, and duration of the decision making task. This models the notion that the time required to make the decision increases as more products are compared, but, that the decision process benefits in terms of quality by having more products.

TÆMS also supports modeling of tasks that arrive at particular points in time, individual deadlines on tasks, earliest start times for tasks, and non-local tasks (those

[3]The actual information gathering task structure does not incorporate outcomes at the task level. This example is a conceptual abstraction of the class of task structures produced by the agent's planner and is simplified for example purposes. Outcomes at the task level have semantics that are difficult to specify.

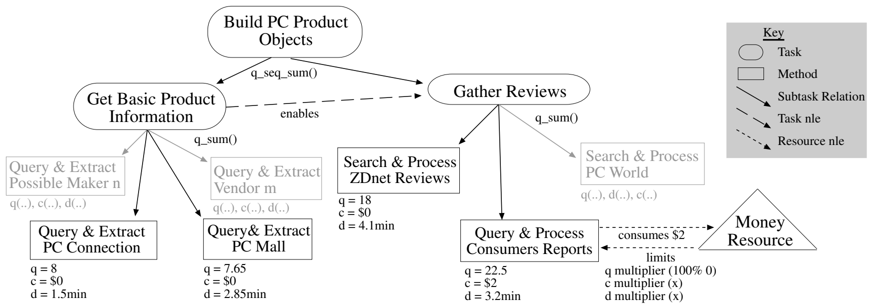


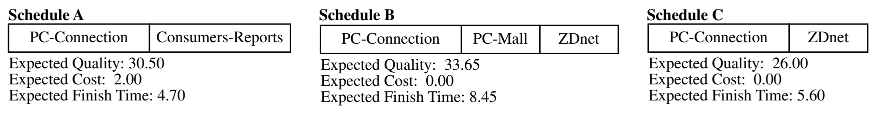
Figure A.2. Simplified Subset of an Information Gathering Task Structure

belonging to other agents). Obviously, scheduling TÆMS task structures is a non-trivial process. In the development of TÆMS there has been a constant tension between representational power and the combinatorics inherent in working with the structure. The result is a model that is non-trivial to process and schedule in any optimal sense, but also one that lends itself to flexible and approximate processing strategies.

Design-to-Criteria Scheduling

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 quality 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. The scheduling problem is exponential and complicated by the existence of task interactions, i.e., primitive actions may not be independent, and by the existence of individual constraints on the primitive actions, e.g., individual deadlines, cost limits, earliest start times, and quality requirements. The combinatorics of the scheduling problem are controlled through the use of approximation, satisficing, goal-directed problem solving, and heuristics for action ordering, as discussed in [82].

The Design-to-Criteria scheduling problem is framed in terms of a TÆMS [18, 83] task network, which imposes structure on the primitive actions and defines how they are related. The most notable features of TÆMS are its domain independence, the explicit modeling of alternative ways to perform tasks, the explicit and quantified modeling of interactions between tasks, and the characterization of primitive actions in terms of quality, cost, and duration. TÆMS is described in greater detail in Section A, however, to ground further discussion consider the TÆMS task structure shown in Figure A.2. The task structure is a conceptual, simplified sub-graph of a task structure emitted by the BIG [43] information gathering agent; it describes


Schedule A
PC-Connection | Consumers-Reports
Expected Quality: 30.50
Expected Cost: 2.00
Expected Finish Time: 4.70

Schedule B
PC-Connection | PC-Mall | ZDnet
Expected Quality: 33.65
Expected Cost: 0.00
Expected Finish Time: 8.45

Schedule C
PC-Connection | ZDnet
Expected Quality: 26.00
Expected Cost: 0.00
Expected Finish Time: 5.60

Figure A.3. Different Schedules Produced for Different Design Criteria

a portion of the information gathering process. The top-level task is to construct product models of retail PC systems. It has two subtasks, *Get-Basic* and *Gather-Reviews*, both of which are decomposed into primitive actions, called *methods*, that are described in terms of their expected quality, cost, and duration. The *enables* arc between *Get-Basic* and *Gather* is a non-local-effect (nle) or task interaction; it models the fact that the review gathering methods need the names of products in order to gather reviews for them. *Get-Basic* has two methods, joined under the *sum()* quality-accumulation-function (*qaf*), which defines how performing the subtasks relate to performing the parent task. In this case, either method or both may be employed to achieve *Get-Basic*. The same is true for *Gather-Reviews*. The qaf for *Build-PC-Product-Objects* is a *seq_sum()* which indicates that the two subtasks must be performed, in order, and that their resultant qualities are summed to determine the quality of the parent task; thus there are nine alternative ways to achieve the top-level goal in this particular sub-structure.

Three different schedules for achieving the top-level goal of the task structure, produced for three different sets of design criteria, are shown in Figure A.3. Schedule A is constructed for a client who needs a high quality solution, requires the solution in seven minutes or less, and who is willing to pay for it. Schedule B is constructed to suit the needs of a client who has plenty of time and is willing to wait for a high quality solution, but who also has no money. Schedule C is constructed for a client who has neither time nor money. Even this example illustrates the notion of quantified choice in TÆMS and how the Design-to-Criteria methodology leverages the quantification to build different schedules for different contexts. However, this simple example also illustrates a weakness in TÆMS as presented in Figure A.2 – a weakness that is carried forward to the scheduling process and consequently to the schedules returned to the client. The initial design of TÆMS included only expected value modeling of primitive actions and task interactions. Subsequently, the authors have understood the strength of explicit modeling of uncertainty and the implications of these new models to the Design-to-Criteria scheduling process.

Prior to delving into an intellectual discussion of the role of uncertainty, consider the simplified task structure revised to include uncertainty, Figure A.4, in the characterizations of the primitive actions. In the enhanced task structure, primitive actions are characterized statistically via discrete probability distributions rather than expected quality values. The quality distributions model the probability of obtaining different quality results and the possibility of failure (indicated by a zero quality result). Note that the expected values of these distributions are the same as those in the previous expected-value model, thus the structures are directly comparable. The

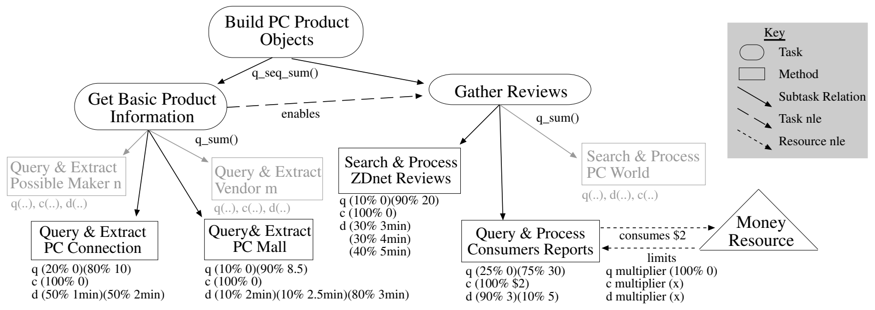
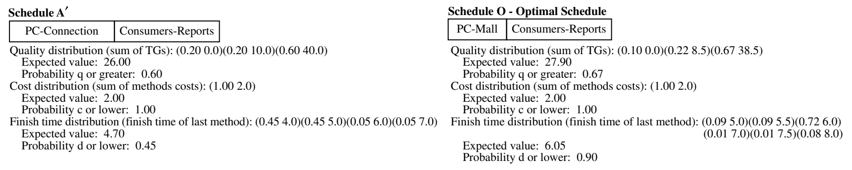


Figure A.4. Simplified Subset of an Information Gathering Task Structure



```
Schedule A′
| PC-Connection | Consumers-Reports |
Quality distribution (sum of TGs): (0.20 0.0)(0.20 10.0)(0.60 40.0)
   Expected value:  26.00
   Probability q or greater:  0.60
Cost distribution (sum of methods costs): (1.00 2.0)
   Expected value:  2.00
   Probability c or lower:  1.00
Finish time distribution (finish time of last method): (0.45 4.0)(0.45 5.0)(0.05 6.0)(0.05 7.0)
   Expected value:  4.70
   Probability d or lower:  0.45
```

```
Schedule O - Optimal Schedule
| PC-Mall | Consumers-Reports |
Quality distribution (sum of TGs): (0.10 0.0)(0.22 8.5)(0.67 38.5)
   Expected value:  27.90
   Probability q or greater:  0.67
Cost distribution (sum of methods costs): (1.00 2.0)
   Expected value:  2.00
   Probability c or lower:  1.00
Finish time distribution (finish time of last method): (0.09 5.0)(0.09 5.5)(0.72 6.0)
                                                       (0.01 7.0)(0.01 7.5)(0.08 8.0)
   Expected value:  6.05
   Probability d or lower:  0.90
```

Figure A.5. Uncertainty Representation Changes Optimal Schedule

cost and duration distributions represent the different possible costs and durations of the actions. This level of detail can be very important when reasoning about the gathering process. For example, in the enhanced model, it is clear that the method for querying and extracting text obtained from the *PC-Connection* site has a higher probability of failure than the method for querying and extracting text obtained from the *PC-Mall* site. In the original model, the detail is lacking and it is impossible to ascertain which method is more likely to fail.

The schedules shown in Figure A.5 illustrate the value of uncertainty in this model from a scheduling perspective. Schedule A′ is identical to Schedule A from the expected value case (Figures A.2 and A.3), however, with the addition of uncertainty to the model, the scheduler can propagate uncertainty and create better estimates for the performance characteristics of the schedules. Note that the quality distribution for Schedule A′ includes a 20% chance of failure. In fact, with the addition of uncertainty to the model, analysis shows that Schedule A is no longer the optimal schedule for the client (who needs a result in 7 minutes or less and is willing to pay for it). Instead Schedule O (Figure A.5) is the optimal choice. Even though the *PC-Connection* method has a higher expected value, the *PC-Mall* method has a lower probability of failure. Since a failure in one of these methods precludes the execution of *Query-Consumers-Reports* (via the task interaction), the issue of failure is not local to the methods but instead impacts the schedule as a whole. Thus, when uncertainty is modeled and propagated during the scheduling process, Schedule O is the optimal schedule as it has the highest net expected quality value and it still meets the client's deadline constraint.

This example conceptually illustrates one aspect of the value of uncertainty in the task models and in the scheduling process – better models lead to better schedules. The authors believe that representing and reasoning about uncertainty is one of the keys to scheduling computational structures in uncertain environments. This is particularly true when quality requirements and time and cost constraints are present. Additionally, with the inclusion of uncertainty modeling and propagation it is clear that there are many different dimensions and aspects of utility that can be used to evaluate the appropriateness of schedules. Consider the task of gathering information via the highly uncertain WWW to support a decision about the purchase of a new automobile. Certain clients may prefer a risky information gathering plan that has a potentially high pay-off in terms of information gathered, but also has a high probability of failure. Other, more risk averse clients might prefer a course of action that results in a lower pay-off in exchange for more certainty about the pay-off and a lower probability of failure. Integrating notions of uncertainty in to the schedule evaluation process is one aspect of this work.

This is work by Tom Wagner.

MASS - Multi Agent Survivability Simulator

The Multi Agent Survivability Simulator (MASS) provides a concrete, re-runnable, well-defined environment to test multi-agent coordination/negotiation. The goal is to provide a distributed simulation system to test various coordination mechanisms allowing the elements of the system to detect, react and adapt to adverse working conditions. The assumption is that the system is composed of a group of autonomous agents. Each agent has its own local view of the world and its own goals, but is capable of coordinating these goals with respect to remote agents.

The simulator is designed as a central process; all agents involved with the model are connected to the simulator using sockets. The agents themselves are independent processes, which could run on physically different machines. Time synchronization is controlled by the simulator, which periodically sends a pulse to each of its remote agents. Each agent has a local manager which converts the simulation pulse into real CPU time. The manager controls the process and records the time spent scheduling, planning, or executing methods. Note that the simulator does not control the agents' activities, it merely allocates time slices and records the activities performed by the agent during the time slice via an instrumentation interface. Once the alloted CPU time has been used, the manager halts the agent and sends an acknowledgment message back to the simulator. The simulator waits for all acknowledgments to arrive before sending a new pulse (so all processes are synchronized). Using this method, any functions (even computationally long ones) can be executed using whatever pulse-granularity is desired (i.e., if one wishes to simulate an agent environment where all planning actions are completed in a single time pulse, the pulse-granularity is defined by the longest running planning action). For example, a planning phase may explore a very large space search, but the simulated time required may be just one pulse. Since all remote processes are frozen until the planning is completed, this gives the illusion that the planning was actually performed very quickly. Each agent therefore has its own task-specific time transformation function, which it uses to convert each simulator pulse into local CPU time. However, the clock mechanism works at the other end of the spectrum, too. If one wishes to study actual planning times, where slower planners take longer, or agents executing on slower machines take longer, then the pulse-granularity is smaller and a tick may correspond to a second in real time or CPU time.

The simulator is also a message router, in that all agent communications will pass through it. This scheme permits explicit control over network and communication delays. In this way, if one wants to simulate a very fast communication path, the simulator may immediately re-send a message to its destination; but if one wants to simulate a compromised network, the simulator may wait n pulses before sending the message. This method also allows an agent to broadcast a message to all other agents without explicitly knowing the number of agents that will receive the message.

The simulator behavior is directed by a queue containing a time-ordered list of events. Each message it receives either adds or removes events from the queue. At each pulse the simulator selects the correct events and realizes their effects (for example, a network may slow down). Only after the effect of each event has been completely determined is the timing pulse sent to each agent. Primitive actions in TÆMS called methods,are characterized statistically via discrete probability distributions in three dimensions: quality, cost, and duration. Agents reason about these characteristics when deciding which actions to perform and at what time.

When an agent wants to execute a method, it sends a message to the simulator with the method name. The simulator then retrieves the method from the objective TÆMS database. Agents schedule, plan, and interact using a subjective view of the methods. The subjective view differs from the objective view because agents may have an imperfect model of what will actually happen, performance-wise, when the method is executed. For example, for a method that involves retrieving data from a remote site via the WWW, the remote site's performance characteristics may have changed since the agent learned them and thus the agent's view of the execution behavior of that method, namely its duration, is imperfect. In a simulation environment, both the subjective and the objective method views are created by the simulator / task generator and the objective, or true, views of the methods are stored in the simulator's TÆMS database. Thus when an execution message arrives, the simulator must obtain the objective view of the method before any other steps may be taken. The first step of the simulator is to calculate the value of the cost, duration and quality that will be "produced" by this execution. The duration (in pulse time) is used to create an event that sends to the agent the results in terms of cost and quality. Any event realized before the newly queued event is performed may change the results of the newly queued event's "execution." For example, a network breakdown event at the front of the queue may increase the time of all the simulated executions that follow it in the queue, by 100%. Thus subsequent method completion events are delayed.

This interaction effect is also possible because of the interactions between the methods themselves. For example, if one method enables another method, and the first method fails, then the other method may no longer be executed or executing the method will produce no result. If both methods are already "scheduled" in the event queue, and the first method fails, then the event associated with the second method's execution must be changed.

The random generator used to calculate the cost, duration and quality values is seeded by a fixed parameter or by a random number (depending on the current time). The solution of seeding by a fixed parameter is used to in order to obtain a deterministic simulation. Our random generator produces the same answer if the same seeding value is used. The goal is to compare different agent coordination mechanisms on the same problem using the same simulation. To test a particular coordination mechanism on several problems, the authors use a seeding based on the current time that guarantees that two simulations do not produce the same solution.

This simulator was designed and implemented by Regis Vincent and Bryan Horling.

JAF - The Java Agent Framework

Java Agent Framework (JAF) is a component-based framework to define agents working within the Mass environment. It effectively isolates the agent-dependent behavior logic from the underlying support code which would be common to all of the agents in the simulation. One goal of the framework is therefore to allow an agent's behavioral logic to perform without the knowledge that it was operating under simulated conditions, e.g. a problem solving component in a simulated agent would be the same as in a real agent of the same type. The framework is also flexible and extensible, and yet maintains separation between mutually dependent functional areas to the extent that one could be replaced without modifying the other.

Component based architectures are a relatively new arrival in the field of software engineering which build upon the notion of object-oriented design. They attempt to effectively encapsulate the functionality of an object while respecting interface conventions, thereby enabling the creation of stand alone applications by simply plugging together groups of components. This paradigm is ideal for our agent framework, because it permits the creation of a number of common-use components, which other agent-dependent components can easily make use of.

JAF is based on Java Beans, Sun Microsystem's component model architecture. Java Beans supplies JAF with a set of design conventions, which provides behavior and naming specifications that every component must adhere to. Specifically, the Java Beans API gives JAF a set of method naming and functional conventions which allow both application construction tools and other beans to easily manipulate a component's state and make use of its functionality. JAF also makes heavy use of Java Bean's notion of event streams, which permit dynamic interconnections to form between stream generating and subscribing components.

JAF builds upon the Java Beans model by supplying a number of facilities designed to make component development and agent construction simpler and more consistent. Mechanisms are provided to specify and resolve both data and inter-component dependencies. These methods allow a component, for instance, to specify that it can make use of a certain kind of data if it is available, or find and make use of other components in the agent. More structure has also been added to the execution of components by breaking runtime into distinct intervals (e.g. initialization, execution, etc.) with associated behavioral conventions during these intervals.

To date, more than 27 JAF components have been built, ranging from simple logging tools to a contract-net based mobile robot problem solver. A sample of these are:

1. Control : Provides initialization facilities and runtime state conventions and control.

2. State : Serves as a central repository for data storage and retrieval within the agent. Also assists with component discovery.

3. Log : Provides multi-level logging facilities.

4. Execute : A generic interface for enabling real or simulated behaviors.

5. Communicate : Supports multiple incoming or outgoing TCP streams, with heterogenous message encodings.

6. PreprocessTaemsReader : Wraps the taems preprocessor, which uses a marked up version of textual Taems to produce task structures for the agent.

7. Observe Simple periodic or event-based engine allowing statistics to be gathered on the agent's internal activities.

8. LogViewer An extensible log visualization tool which can show a consolidated, multi-agent view of agent activities in real time.

This framework was defined and implemented by Bryan Horling.

APPENDIX B

ALTERNATE FORMAL DEFINITION

The following is an alternate formal definition to the meta-level control problem. A decision theoretic definition of the problem was presented in Chapter 1.

The goal of the meta-level controller is to determine an optimal control policy for intermixing domain and control activities for the set of tasks over a given horizon. The decisions include determining whether, when and how to execute each of the domain tasks when there is the option to spend more or less time on control or reasoning about these activities.

An agent receives external requests to achieve goals.

1. G_i is a goal of a specific type **i**.
2. G_{i_k} is the **kth** goal received by the agent.
3. When a goal is received by an agent, it is added to the **new goal list**. $NGLIST(t)$ is the set of goals on the new goal list at time t. It is constructed dynamically; a goal is added when it arrives into the system and it is removed when it is reasoned about by the meta-level controller. A goal which is removed can be dropped, added to a waiting list called agenda or can be sent to the control component for scheduling. Multiple goals in the new task list are reasoned about sequentially.
4. G_{i_k} has an associated **arrival time** $AT(G_{i_k})$, an externally specified **earliest start time** $EST(G_{i_k})$ and a **deadline** $DL(G_{i_k})$.
5. An agent at any instant of time t has an **agenda** of goals. An agenda is constructed dynamically; an old goal is removed when it is scheduled for execution or a decision to drop it is made, and a new goal is added to the set when a decision to do so is made. There is a probabilistic model of goal arrivals for each agent. $AGENDA(t)$ is the set of goals on the agenda at time t. These are goals which have arrived at the agent but are yet to be reasoned about and scheduled.

A **domain activity** is a procedure for solving a goal.

1. For each goal G_i, there are n_i alternative domain activities which can solve it. T_i^j is the **jth** alternative domain activity which which can solve goal G_i.

2. Each alternative T_i^j has an **expected utility** $EU(T_i^j)$ and **expected duration** associated with it $ED(T_i^j)$.

3. $T_{i_k}^{a_k}$ represents the domain activity that is chosen to solve goal G_{i_k}.

4. The domain activities which have a higher expected duration, have a higher expected utility. The alternatives are ordered in increasing order of duration.

$$\forall i, j \leq k, ED(T_i^j) \leq ED(T_i^k) \wedge EU(T_i^j) \leq EU(T_i^k)$$

5. $ACTIVE(t)$ is the set of domain activities currently executing as well as those which have been scheduled and are waiting to be executed at time t.

6. A domain activity $T_{i_k}^{a_k}$ which is in $ACTIVE(t)$ at time t has a **starting time** $ST(T_{i_k}^{a_k})$ and a **expected finishing time** $EFT(T_{i_k}^{a_k})$.

7. When a particular alternative T_i^j for a goal is executing, $D_{used}(T_{i_k}^{a_k}, t)$ is the **total processor time** used by the alternative at time t, $U_{acc}(T_{i_k}^{a_k}, D_{used}(T_{i_k}^{a_k}, t))$ is the **utility accrued** for that duration and $D_{left}(T_{i_k}^{a_k}, t)$ is the **expected time left to complete execution** of the alternative at time t.

8. When an alternative for the goal has completed execution, its **finish time** is denoted $FT(T_{i_k}^{a_k})$, duration in terms of processor time taken by alternative at finish time is $D_{used}(T_{i_k}^{a_k}, FT(T_{i_k}^{a_k}))$ and the utility accumulated by the alternative at the finish time is $U_{acc}(T_{i_k}^{a_k}, FT(T_{i_k}^{a_k}))$.

9. Execution of a domain activity is a function which requires as input a set of domain activities along with their start times and expected finish times. Its output is the finish time and the utility accrued at that finish time for each activity. $EXECUTE\_DTASKS(\{< T_{i_k}^{a_k}, ST(T_{i_k}^{a_k}), EFT(T_{i_k}^{a_k}) >\})$ is denoted by $OUTPUT\_DTASKS$

   Suppose $D1 \; = \; D_{used}(T_{i_k}^{a_k}, FT(T_{i_k}^{a_k}))$

$$OUTPUT\_DTASKS = \{< D1, U_{acc}(T_{i_k}^{a_k}, D1), FT(T_{i_k}^{a_k}) >\}$$

10. If at time t, $T_{i_k}^{a_k} \epsilon ACTIVE(t)$ and $ST(T_{i_k}^{a_k}) \leq t$, then

$$EXECUTE\_DTASKS(\{< T_{i_k}^{a_k}, ST(T_{i_k}^{a_k}), EFT(T_{i_k}^{a_k} >\}))$$

   is executed until interrupted.

11. Domain activities are interleaved by the execution component with control activities.

12. Domain activities can be interrupted by control activities.

A **control activity** determines whether, how and when domain activities can be performed. Control activities also have alternative ways in which they can be accomplished. Suppose there are N different alternatives to complete a control activity in an environment. C^n represents an alternative for completing the control activity where $0 \leq n \leq N-1$

1. Each alternative C^n has a **expected duration** associated with it $ED(C^n)$.
2. Control activities do not produce utility. However, they are necessary for generating the sequence of domain activities which on execution produce utility.
3. Alternatives for a control action vary in their expected durations and ordered by increasing values of their duration.

$$\forall p \leq q \leq N-1, ED(C^p) \leq ED(C^q)$$

4. Suppose the total number of control activities executed by an agent is M. A control activity that is in execution is represented as C_m^n and it stands for the **nth** alternative of the **mth** control activity such that $0 \leq m \leq M-1$.
5. When a particular alternative C_m^n for an activity is executing, $D_{used}(C_m^n, t)$ is the **total processor time** used by the alternative at time t.
6. $SCHEDULED\_TASKS(C_m^n)$ is the set of domain activities which are scheduled as a result of executing the alternative for control activity C_m^n.
7. Control activities are non-interruptible.
8. Execution of a control activity is a function which requires three inputs: the first is the alternative of the control activity; the second is the set of goals to be reasoned about along with the earliest start times and deadline of each goal; the third input is the set of tasks which are currently in execution or are waiting to be executed along with the utility that each of these tasks has accrued up to that point in time as well the expected remaining time left to complete each of these tasks.

$$EXECUTE\_CTASK(INPUT) => OUTPUT$$

where

$$\begin{aligned} INPUT = C_m^n, \\ &\{\forall G_{i_k} \in NGLIST(t) \vee AGENDA(t), < G_{i_k}, EST(G_{i_k}), DL(G_{i_k}) >\}, \\ &\{\forall T_{i_k}^{a_k} \in ACTIVE(t), < T_{i_k}^{a_k}, U_{acc}(T_{i_k}^{a_k}, t), D_{left}(T_{i_k}^{a_k}, t) >\} \end{aligned}$$

Its output can be to **schedule** the goals, **reschedule** the tasks, **delay** reasoning about the goal by adding it to agenda or **drop** the goal completely.

$$OUTPUT = \begin{cases} < OUTPUT1 >, \ \ schedule\ goal\ if\ G_{i_k}\ \in\ AGENDA(t) \\ < OUTPUT2 >, \ \ reschedule\ goal\ if\ G_{i_k}^{a_k}\ \in\ ACTIVE(t) \\ < DELAY(G_k) >, \ \ remove\ from\ NEWGOALLIST(t) \\ and\ add\ to\ AGENDA(t) \\ < DROP(G_k) >, \ \ remove\ goal\ from\ AGENDA(t) \end{cases}$$

$$OUTPUT2 = Remove\ G_{i_k}\ from\ AGENDA(t) \wedge$$
$$Add\ \ < T_{i_k}^{a_k}, ST(T_{i_k}^{a_k}), EFT(T_{i_k}^{a_k}) >\ \ to\ ACTIVE(t))$$

$$OUTPUT3 = Remove < T_{i_k}^{a_k}, ST(T_{i_k}^{a_k}), EFT(T_{i_k}^{a_k}) >$$
$$from\ ACTIVE(t) \wedge \}$$
$$Add < T_{i_k}^{a'_k}, ST(T_{i_k}^{a'_k}), EFT(T_{i_k}^{a'_k}) >\ \ to\ ACTIVE(t)$$

9. If at time t, $ACTIVE(t) = \phi$ and $G_{i_k} \epsilon AGENDA(t)$ and $EST(G_{i_k}) \leq t$, then

$$EXECUTE\_CTASK(C_m^n, \{G_{i_k}, EST(G_{i_k}), DL(G_{i_k})\})$$

is executed.

A **single agent scenario** is a sequence of goals arriving at an agent with each goal G_k^i arriving at a specific time $AT(G_k^i)$, with an associated earliest start time $EST(G_k^i)$ and a deadline $DL(G_k^i)$.

A **multi-agent scenario** is one that has multiple agents with each having its own scenario. Goals can arrive from the external environment or can be transferred from another agent in the multi-agent system.

An **ensemble of scenarios** is the set of multi-agent scenarios which are produced from a set of goals which have a statistical distribution of their arrival times and deadlines.

The **optimization problem** for a given agent in a multi-agent scenario is to maximize the total utility obtained from completing the goals given a set of four constraints. Each constraint is represented by ζ_i. Suppose there are N goals which arrive at an agent over a time horizon $D_{horizon}$.

$$max|\sum_{k=1}^{N} U_{acc}(T_{i_k}^{a_k}, D_{used}(T_{i_k}^{a_k}, FT(T_{i_k}^{a_k}))) : \zeta_1 \wedge \zeta_2 \wedge \zeta_3 \wedge \zeta_4|$$

ζ_1 ensures that the total duration taken to execute the domain activities and control activities is less than the time horizon for the given scenario.

$$\zeta_1 = [\sum_{k=1}^{N} D_{used}(T_{i_k}^{a_k}, FT(T_{i_k}^{a_k})) + \sum_{k=1}^{N} D_{used}(C_m^n, FT(C_m^n)) \leq D_{horizon}]$$

ζ_2 ensures that a domain activity completes before the deadline for that goal and that that deadline is within the scenario's horizon.

$$\zeta_2 = [\forall k, FT(T_{i_k}^{a_k}) \leq DL(G_{ik}) \leq D_{horizon}]$$

ζ_3 ensures that domain activity does not begin execution before its earliest starting time.

$$\zeta_3 = [\forall k, ST(T_{i_k}^{a_k}) \geq EST(T_{i_k}^{a_k})]$$

ζ_4 ensures that the control activity involved in scheduling completes before any affected domain activities begin execution.

$$\zeta_4 = [\forall k, T_{i_k}^{a_k} \in SCHEDULED\_TASKS(C_m^n),\ FT(C_m^n) \leq ST(T_{i_k}^{a_k})]$$

Another way of viewing this problem is as follows:

The dynamic arrival of new goals $G_{i_1}, G_{i_2}...G_{i_k}$ at the agent leads to the creation of an execution sequence involving both domain and control activities that optimize the given function under the constraints. Suppose

$$G^s \epsilon Set\ of\ all\ possible\ orderings\ of\ G_{i_1}, G_{i_2}...G_{i_k}$$

for a given ensemble of goal arrivals and $Pr(G^s)$ is the likelihood that sequence G^s will occur. Then the optimization can be defined in terms of finding the legal execution sequence which optimizes the given function and the meta-level control policy P is the policy for generating a nearly-optimal legal sequence of control activities interspersed with domain activities which are the affectees of the control activities.

Meta-level control is the decision problem of choosing the set of control activities which solve the above optimization problem. This involves selecting the ordering control activities from all such orderings possible in the environment which satisfy the constraints.

Reinforcement Learning is the methodology used in this work to determine the policy which optimizes the above function for an ensemble of scenarios. The state representation for a naive reinforcement learning problem would require information on each of the input variables in the above constraints for each instant in time. This would lead to an extremely large and complex search space. This is infeasible to learn and store. I introduce **bounded rationality** to this meta-level control problem by approximating these variables in the state space so that approximately-optimal policies are produced by the reinforcement learning process while the size of the search space is bounded.

To elucidate the discussion further, consider the simple environment consisting of 2 agents described in the dissertation. For the sake of simplicity, the agents are named A and B. The discussion will specifically focus on agent A. Suppose T_0 and T_1 are the goals achieved by agent A. Each top-level goal is decomposed into executable primitive actions. In order to achieve the goal, agent A can execute one or more of its primitive actions within the goal deadline and the quality accrued for the goal will be cumulative. Figure B.1 describes the alternative ways of achieving each of these goals. Each alternative for goal G_k is called T_k^{ak} and contains information on the sequence of primitive actions representing the alternative as well as the expected utility $EU(T_k^{ak})$ and the expected duration $ED(T_k^{ak})$ of each alternative. Figure B.2 describes the alternative ways of achieving each control activity C_m. Each alternative C_m^n contains name of alternative, a description of the alternative and the expected duration $ED(C_m^n)$ for each alternative.

Alternative	Method Sequence	Expected Utility	Expected Duration
T_0^1	{M1}	6	8
T_0^2	{M2}	10.2	10.2
T_0^3	{M1,M2}	16.2	18.2
T_1^0	{M3}	12	8
T_1^1	{MetaNeg,NegMech1,M4}	12.6	16.2
T_1^2	{MetaNeg,NegMech2,M5}	13.05	16.6
T_1^3	{MetaNeg,NegMech1,M3,M4}	24.6	24.2
T_1^4	{MetaNeg,NegMech2,M3,M4}	25.05	24.6

Figure B.1. Alternative information for domain activities

Consider a scenario where the arrival model for agent A (denoted $GoalName < ArrivalTime, Deadline >$) is as follows:

1. $G_0 < AT = 1, DL = 40 >$,
2. $G_1 < AT = 15, DL = 80 >$,
3. $G_0 < AT = 34, DL = 90 >$,
4. $G_1 < AT = 34, DL = 90 >$.

The earliest start time for the goals in this scenario is the same as its arrival time.

Alternative	Description	Expected Duration
C_m^0	Add goal to agenda and delay reasoning	0
C_m^1	Call detailed scheduler	1
C_m^2	Call abstract scheduler	1
C_m^3	Schedule with no negotiation option	2
C_m^4	Schedule with NegMech1	3
C_m^5	Schedule with NegMech2	3

Figure B.2. Table of possible control activities

The following is the time line which describes agent *A's* activities over a horizon of 100 time units ($D = 100$).

Time 1: Goal G_{0_1} arrives with $EST(G_{0_1}) = 1$ and $DL(G_{0_1}) = 40$).

Time 2: The meta-level controller is invoked and the control activity $EXECUTE\_CTASK(C_1^1, \{< G_{0_1}, 1, 40 >\}, \phi)$ is prescribed as the best action.

Time 3: Control activity $EXECUTE\_CTASK(C_1^1, \{< G_{0_1}, 1, 40 >\}, \phi)$ begins execution.

Time 7: Control activity completes execution and its output is $EXECUTE\_DTASK(\{< T_{0_1}^3, 5, 25 >\})$. The domain activity $EXECUTE\_DTASK(\{< T_{0_1}^3, 5, 25 >\})$ produces the legal schedule **{M1, M2}** and begins execution by initiating execution of primitive action **M1**.

Time 13: Execution of the method **M1** completes with quality of 6. Execution of **M2** which is next on the schedule begins.

Time 16: Goal G_{1_2} arrives with $EST(G_{1_2}) = 15$ and $DL(G_{1_2}) = 80$). Execution of method **M2** is interrupted and control switches from the execution component to the meta-level control component. The meta-level controller is invoked and the control activity $EXECUTE\_CTASK(C_2^0, \{< G_{1_2}, 15, 80 >\}, \{< T_{0_1}^3, 6, 10 >\})$ is prescribed as the best action. Upon execution, the control activity has $< DELAY(G_{1_2}) >$ as its output. So the new goal is added to the agenda and execution of method **M2** is resumed.

Time 26: Method **M2** completes with utility of 12 and goal G_{0_1} completes with finish time $FT(T_{0_1}^3) = 26$ and utility of $U_{acc}(T_{0_1}^3, 18) = 18$. The agenda is checked and goal $< G_{1_2}, 15, 80 >$ is retrieved. The meta-controller is invoked and the control activity $EXECUTE\_CTASK(C_3^5, \{< G_{1_2}, 15, 80 >\}, \phi)$ is prescribed as the best action. The meta-level controller prefers alternative with **NegMech2** in it.

Time 27: Control activity $EXECUTE\_CTASK(C_3^5, \{< G_{1_2}, 15, 80 >\}, \phi)$ begins execution.

Time 31: Control activity completes execution and its output is $EXECUTE\_DTASK(\{< T_{1_2}^4, 29, 80 >\})$. The legal schedule is **{MetaNeg, NegMech2, M3, M4}**. The domain activity begins execution by initiating executing **MetaNeg** and **M3** in parallel.

Time 33: Execution of **MetaNeg** completes. The information gathered by **MetaNeg** is as follows: the other agent is executing schedule with expected utility of 9 and the expected time of completion of that schedule is 80. There is also high slack and high uncertainty in the non-local schedule. The meta-level controller does not prescribe any change to the currently legal schedule. Method **NegMech2** begins execution

Time 34: Goals G_{0_3} and G_{1_4} arrive with $EST(G_{0_3}) = 34$, $DL(G_{0_3}) = 90$, $EST(G_{1_4}) = 34$ and $DL(G_{1_4}) = 90$. Execution of method **M3** and **NegMech2** is interrupted and control switches from the domain-level control component to the meta-level control component. The meta-level controller prescribes the following action: $EXECUTE\_CTASK(C_4^5, \{<G_{0_3}, 34, 90>, <G_{1_4}, 34, 90>\}, \phi)$

Time 36: Control activity $EXECUTE\_CTASK(C_4^5, \{<G_{0_3}, 34, 90>, <G_{1_4}, 34, 90>\}, \{<T_{1_2}^4, 1, 21>\})$ begins execution.

Time 43: Control activity completes execution and its output is $EXECUTE\_DTASK(\{<T_{0_3}^3, 38, 90>, <T_{1_2}^4, 29, 80>\}), DROP(G_{1_4})$. The legal schedule emitted is **{NegMech2, M3, M4, M1, M2}** The domain activity begins execution by initiating execution of **NegMech2** in parallel with continuing execution of method **M3**.

Time 46: Method **NegMech2** completes successfully with a commitment for method **M4** will be enabled at time 65. Control shifts to the meta-control component which now decides whether to renegotiate by initiating $EXECUTE\_CTASK(C_5^0, \phi, \{<T_{0_3}^3, 0, 20>, <T_{1_2}^4, 1, 16>\})$. method **M4** will be enabled at time 65.

Time 52: Method **M3** completes with utility 12.

Time 58: Execution of **M4** completes with a successful commitment: method **M4** will be enabled at time 65. Execution of **M1** on the schedule begins.

Time 64: Execution of method **M1** completes with a utility of 6. Execution of **M2** begins.

Time 65: Method **M4** is enabled by non-local agent. Execution of **M2** is interrupted and execution of **M4** begins.

Time 77: Execution of method **M4** completes with a utility of 12 and goal G_{1_2} completes with $FT(T_{1_2}^3) = 77$ and utility of $U_{acc}(T_{1_2}^3, 34) = 25$. Execution of **M2** is resumed.

Time 80: Execution of **M2** completes. Execution of goal G_{0_3} completes with $FT(T_{0_3}^3) = 88$ and $U_{acc}(T_{0_3}^3, 18) = 18$ completes. $U_{total} = 58$. The agenda is checked and ϕ is returned since agenda is empty.

APPENDIX C

AGENT *FRED'S* EXECUTION-TIME BEHAVIOR

The following is a system trace for a particular execution scenario for *Fred*. The trace describes the actual state of the agent along a time-line and the corresponding meta-level decisions that are made by the agent.

State S1:

CurrentTime : 2
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
MLC Decision : **Call Detailed Scheduler**

State S2:

CurrentTime : 3
NewTaskList : ϕ ; AgendaList : ϕ
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 Scheduler are EffortLevel=2; Slack=10%**

State S3:

CurrentTime :5
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList : $\{GettoRockLocation, FocusSpectrometeronRock\}$
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 : **Begin Execution of GettoRockLocation**

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 : **Begin Execution of FocusSpectrometeronRock**

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**

State S6:
CurrentTime :17
NewTaskList : ϕ; AgendaList : $ExploreTerrain < 15, 80 >$
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 : **Resume execution of interrupted method FocusSpectrometeronRock**

State S7:
CurrentTime :25
NewTaskList : ϕ; AgendaList : $ExploreTerrain < 15, 80 >$
ScheduleList: ϕ; ExecutionList : ϕ
InformationGathered : ϕ
Utility of current schedule : 18.0; Duration of current schedule : 18.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 18.0
MLC Decision : **Evaluate task in agenda**

State S8:
CurrentTime :26
NewTaskList : ϕ; AgendaList : $ExploreTerrain < 15, 80 >$
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 : 18.0
MLC Decision : **Call detailed scheduler**

State S9:
CurrentTime :28

NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: $ExploreTerrain < 15, 80 >$; 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 : 18.0
MLC Decision : **Parameters for Scheduler are EffortLevel=2; Slack=30%**

State S10:
CurrentTime :30
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{MetaNeg, NegMech2, ExamineTerrain, CollectSamples\}$
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 : 18.0
MLC Decision : **Begin execution of Information Gathering Action (MetaNeg) in parallel with ExamineTerrain**

State S11:
CurrentTime :33
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{NegMech2, ExamineTerrain, CollectSamples\}$
InformationGathered : $< HIGH, LOW, HIGH$
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**

State S12:
CurrentTime :35
NewTaskList : $AnalyzeRock < 34, 90 >, ExploreTerrain < 34, 90 >$; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{NegMech2, ExamineTerrain^{exe}, CollectSamples\}$
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 3.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 18.0
MLC Decision : **Call detailed scheduler on all lists**

State S13:
CurrentTime :37
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: $AnalyzeRock < 34, 90 >, ExploreTerrain < 34, 90 >, ExploreTerrain^{exe} < 15, 80 >$; ExecutionList :$\phi$
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 3.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;

Total Utility accrued : 18.0
MLC Decision : **Parameters for Scheduler are EffortLevel=2; Slack=30%**

State S14:
CurrentTime :40
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ;
ExecutionList :$\{NegMech2, ExamineTerrain^{exe},$
$CollectSamples, GettoRockLocation, FocusSpectrometeronRock\}$
InformationGathered : ϕ
Utility of current schedule : 0.0; Duration of current schedule : 3.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 2.0;
Total Utility accrued : 18.0
MLC Decision : **Execute NegMech2 in parallel with method ExamineTerrain**

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 tume 65. Continue execution of ExamineTerrain**

State S16:
CurrentTime :52
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ;
ExecutionList :$\{CollectSamples, GettoRockLocation, FocusSpectrometeronRock\}$
InformationGathered : ϕ
Utility of current schedule : 9.0; Duration of current schedule : 15.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 27.0
MLC Decision : **Begin execution of GettoRockLocation**

State S17:
CurrentTime :58
NewTaskList : ϕ; AgendaList : ϕ
ScheduleList: ϕ; ExecutionList :$\{CollectSamples, FocusSpectrometeronRock\}$
InformationGathered : ϕ
Utility of current schedule : 15.0; Duration of current schedule : 21.0;
Utility of interrupted action : 0.0; Duration of interrupted action : 0.0;
Total Utility accrued : 33.0
MLC Decision : **Begin execution of FocusSpectrometeronRock**

State S18:

CurrentTime :65

NewTaskList : ϕ; AgendaList : ϕ

ScheduleList: ϕ; ExecutionList :$\{CollectSamples, FocusSpectrometeronRock^{exe}\}$

InformationGathered : ϕ

Utility of current schedule : 21.0; Duration of current schedule : 27.0;

Utility of interrupted action : 6.0; Duration of interrupted action : 6.0;

Total Utility accrued : 39.0

MLC Decision : **Interrupt FocusSpectrometeronRock and Begin execution of CollectSamples**

State S19

CurrentTime :77

NewTaskList : ϕ; AgendaList : ϕ

ScheduleList: ϕ; ExecutionList :$\{FocusSpectrometeronRock^{exe}\}$

InformationGathered : ϕ

Utility of current schedule : 33.0; Duration of current schedule : 39.0;

Utility of interrupted action : 6.0; Duration of interrupted action : 6.0;

Total Utility accrued : 51.0

MLC Decision : **Resume execution of FocusSpectrometeronRock**

State S20

CurrentTime :80

NewTaskList : ϕ; 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 : 57.0

MLC Decision : **Go to wait state**

BIBLIOGRAPHY

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

[2] Bazzan, Ana L.C., Lesser, Victor, and Xuan, Ping. Adapting an Organization Design through Domain-Independent Diagnosis. Computer Science Technical Report TR-98-014, University of Massachusetts at Amherst, February 1998.

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

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

[5] Bonasso, R. Peter. Integrating reaction plans and layered competences through synchronous control. *Proceedings of the Twelvth International Joint Conference on Articial Intelligence* (1991), 1225–1231.

[6] Bonasso, R. Peter, Firby, James, Gat, Erann, Kortenkamp, David, Miller, David P., and Slack, Marc G. Experiences with an architecture for intelligent, reactive agents. vol. 9, pp. 237–256.

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

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

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

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

[11] Carver, Norman, and Lesser, Victor. The DRESUN testbed for research in FA/C distributed situation assessment: Extensions to the model of external evidence. In *Proceedings of the First International Conference on Multiagent Systems* (June, 1995).

[12] Connell, Jonathan H. Sss: A hybrid architecture applied to robot navigation. *Proceedings of IEEE Journal of Robotics and Automation* (1991), 2719–2724.

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

[14] Das, S. The Measurement of Flexibility in Manufacturing Systems. *International Journal of Flexible Manufacturing Systems 8* (1996), 67–93.

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

[16] Decker, K., Sycara, K., and Zeng, D. Designing a multi-agent portfolio management system, 1996.

[17] Decker, Keith, and Li, Jinjiang. Coordinated hospital patient scheduling. pp. 104–111.

[18] Decker, Keith S. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.

[19] Decker, Keith S. TÆMS: A framework for analysis and design of coordination mechanisms. In *Foundations of Distributed Artificial Intelligence*, G. O'Hare and N. Jennings, Eds. Wiley Inter-Science, 1995, ch. 16.

[20] Decker, Keith S., and Lesser, Victor R. Coordination assistance for mixed human and computational agent systems. In *Proceedings of Concurrent Engineering 95* (McLean, VA, 1995), Concurrent Technologies Corp., pp. 337–348. Also available as UMASS CS TR-95-31.

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

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

[23] Doyle, Jon. Artificial intelligence and rational self-government. Computer Science Technical Report CS-88-124, Carnegie-Mellon University Computer Science Department, April 1988.

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

[25] Fehling, M. R., , Breese, J. S, and Horvitz, Eric J. A computational model for decision-theoretic control of problem-solving under uncertainty.

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

[27] Gat, Erann. *Reliable Goal-Directed Reactive Control for Real-World Autonomous Mobile Robots.* PhD thesis, Virginia Polytechnic Institute and State University, 1991.

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

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

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

[31] Harada, Daishi, and Russell, Stuart. 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).

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

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

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

[35] Jr., Stanley M. Sutton, and Osterweil, Leon J. The design of a next-generation process language. In *Proceedings of the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering* (September 1997), pp. 142–158.

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

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

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

[39] Kuwabara, Kazuhiro, Ishida, Toru, and Osato, Nobuyasu. Agentalk: Coordination protocol description for multiagent systems. In *Proceedings of the First International Conference on Multi–Agent Systems* (San Francisco, CA, 1995), Victor Lesser, Ed., MIT Press, p. 455.

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

[41] Laird, J, Newell, A., and Rosenbloom, P. Soar: An architecture for general intelligence. *Artificial Intelligence 33*, 1 (1987), 1–64.

[42] Lesser, V. R., and Corkill, D. D. Functionally accurate cooperative distributed systems. *IEEE trans on systems, machines and cybernetics SMC-11*, 1 (1981), 81–96.

[43] Lesser, Victor, Horling, Bryan, Klassner, Frank, Raja, Anita, Wagner, Thomas, and Zhang, Shelley XQ. BIG: A resource-bounded information gathering agent. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* (July 1998). See also UMass CS Technical Reports 98-03 and 97-34.

[44] Lesser, Victor, Michael, Atighetchi, Benyo, Brett, Horling, Bryan, Raja, Anita, Vincent, Regis, Wagner, Thomas, Xuan, Ping, and Zhang, Shelly XQ. The umass intelligent home project. In *Proceedings of the Third Conference on Autonomous Agents* (1999). http://mas.cs.umass.edu/research/ihome/.

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

[46] Lipman, B. How to decide how to decide how to : : :: Limited rationality in decisions and games. In *Working Notes of the AAAI Symposium on AI and Limited Rationality* (1989.), pp. 77–80.

[47] Littman, Michael, and Boyan, Justin. A distributed reinforcement learning scheme for network routing. Tech. Rep. CS-93-165, 1993.

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

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

[50] Musliner, D. J., Hendler, J. A., Agrawala, A. K., Durfee, E. H., Strosnider, J. K., and Paul, C. J. The Challenges of Real-Time AI. In *IEEE Computer* (1995), pp. 28(1):58–66.

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

[52] Nakakuki, Yoichiro, and Sadeh, Norman. 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)* (1994), pp. 1316–1322.

[53] Oates, T., Nagendra Prasad, M. V., and Lesser, V. R. Cooperative Information Gathering: A Distributed Problem Solving Approach. Computer Science Technical Report 94–66, University of Massachusetts, 1994. Journal of Software Engineering, Special Issue on Developing Agent Based Systems, 1997.

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

[55] Prasad, M V Nagendra, and Lesser, Victor. The use of meta-level information in learning situation specific coordination. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (1997).

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

[57] Raja, Anita, Lesser, Victor, and Wagner, Thomas. Toward Robust Agent Control in Open Environments. In *Proceedings of the Fourth International Conference on Autonomous Agents* (Barcelona, Catalonia, Spain, July, 2000), ACM Press, pp. 84–91.

[58] Russell, S., and Norvig, P. *Articial Intelligence: A Modern Approach.* Prentice Hall, 1995.

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

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

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

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

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

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

[65] Scerri, P., Pynadath, D., and Tambe, M. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the Fifth International Conference on Autonomous Agents(Agents-01)* (Montreal, Canada), ACM Press.

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

[67] Sethi, A, and Sethi, S. Flexibility in Manufacturing: A Survey. *International Journal of Flexible Manufacturing Systems 2* (1990), 289–328.

[68] Shoham, Y., and Tennenholtz, M. Co-learning and the evolution of coordinated multi-agent activity, 1993.

[69] Simon, H. From substantive to procedural rationality. 129–148.

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

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

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

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

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

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

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

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

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

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

[80] Tesauro, Gerald. Practical issues in temporal difference learning. In *Advances in Neural Information Processing Systems* (1992), John E. Moody, Steve J. Hanson, and Richard P. Lippmann, Eds., vol. 4, Morgan Kaufmann Publishers, Inc., pp. 259–266.

[81] Vincent, Regis, Horling, Bryan, Wagner, Thomas, and Lesser, Victor. Survivability simulator for multi-agent adaptive coordination. In *Proceedings of the First International Conference on Web-Based Modeling and Simulation* (1998). To appear. Also available as UMASS CS TR-1997-60.

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

[83] Wagner, Thomas, and Lesser, Victor. Toward Ubiquitous Satisficing Agent Control. In *1998 AAAI Symposium on Satisficing Models* (March, 1998).

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

[85] Weiss, G. Learning to coordinate actions in multi-agent systems. In *Proceedings of IJCAI-93* (Chambery, France, 1993), Morgan Kaufmann, pp. 311–317.

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

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

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

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