The Role of Meta-Reasoning in Achieving Effective Multi-Agent Coordination

Zachary Rubinstein, Stephen F. Smith, and Terry Zimmerman

The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{zbr,sfs,wizim}@cs.cmu.edu

Abstract. In this paper, we consider the role of meta-reasoning in achieving effective coordination among multiple agents in maintaining and executing joint plans in an uncertain environment. We assume that each agent has responsibility for performing a particular set of activities in the plan over time. Each agent is provided with an initial schedule and a set of alternative activities, but no agent has a global view of either the problem or solution. Individual agents must maintain their local schedules as execution events deviate from expectations. In circumstances where there are inter-dependencies among the activities of multiple agents, agents must coordinate. We describe an agent architecture designed to provide this distributed schedule management capability and to frame the meta-level control problem that it presents. We identify several degrees of freedom in configuring the agent’s core computational components, each of which affects the proportion of computational cycles given to local scheduling and inter-agent coordination processes. We motivate the need for on-line reasoning by considering how aspects of the current control state impact the utility of different configurations. We present experimental results that demonstrate the potential benefit of adapting configurations to match the current control state. Finally, we sketch a framework for learning this mapping from experience as multi-agent schedule management problems are solved.

1 Introduction

The ability to dynamically manage internal computational activity is important in many agent-based systems. We focus in this paper on a system of scheduling agents, which are engaged in managing and executing a joint plan in an uncertain environment. In this context, an agent has limited time to take correcting actions in response to unexpected execution results. It must balance the time it spends locally revising its schedule (actions aimed at restoring feasibility and/or capitalizing on detected opportunities for local schedule improvement) with the time it spends coordinating with other agents (actions aimed at identifying and exploiting opportunities for more global schedule improvement through joint change). The former is necessary to keep execution going but may be prone to myopic, suboptimal decisions, given its local incomplete view of the overall problem and solution. The latter can lead to better joint scheduling decisions, but
the computation and communication costs in obtaining them may render them obsolete before they can be acted on. Given these trade-offs, it makes sense to try to exploit aspects of the current control state (e.g., the tempo of execution and level of dynamics in the environment) to dynamically configure appropriate sorts of computational actions.

The specific system of interest in our research is the “CMU Agent” [1], one of three competing approaches under development within the DARPA Coordinators Program for solving the distributed schedule management problem. In brief, this problem requires a set of agents to jointly execute a schedule so as to maximize the quality obtained by all executed activities. These ”scenarios” provide each agent with the portion of the initial schedule that it is responsible for, a set of alternative (substitutable) activities and associated outcome and duration probabilities for all assigned activities. Each agent is also given visibility of inter-dependent activities that have been assigned to other agents, but no agent has a global view of either the problem or solution. The CMU Agent design takes a scheduler-centric perspective to solving the Coordinators problem. An incremental, flexible times scheduler sits at its core and is used to drive the agent’s two core processes: (1) it is invoked to perform local scheduling in response to external feedback, and (2) it is invoked hypothetically to generate and evaluate non-local options in the course of inter-agent coordination.

In the sections below, we consider the control problem that the CMU agent faces in allocating cycles to each of these core processes. In operation, our multi-agent system must interact with a simulated environment in near real-time, and hence only limited computational cycles are available for allocation to either of these processes. We examine the hypothesis that dynamic management of control parameters related to this division of computational effort between local scheduling and inter-agent coordination can lead to improved performance over any fixed configuration of these parameters. Before discussing the control parameters of interest and settling on a specific subset for experimental analysis, we briefly summarize the current CMU Agent.

2 Overview of The CMU Agent

The CMU agent architecture is schematically depicted in Figure 1. In its most basic form, an agent comprises four principal components - an Executor, a Scheduler, a Distributed State Manager (DSM), and an Options Manager - all of which share a common model of the current scenario and solution state. This common model couples a domain-level representation of the agent’s local (subjective) view of the overall scenario (encoded as a c_tamens[2] task structure) to an underlying Simple Temporal Network (STN) [3]. At any point during operation, the currently installed schedule dictates the timing and sequence of domain-level activities that will be initiated by the agent. The Executor, running in its own thread, continually monitors the enabling conditions of various pending activities and activates the next pending activity as soon as all of its causal and temporal constraints are satisfied. The other three components run on a separate thread
in a blackboard-based control regime and have responsibility for coordinating with other agents and managing the current schedule over time.

When execution results are received back from the environment (shown in Figure 1 as the MASS simulator, i.e., the execution simulator provided by the Coordinators program) and/or changes to assumed external constraints are received from other agents, the agent’s model of current state is updated. An incremental propagator based on [4] is used to infer consequences within the STN. In cases where this update leads to inconsistency in the STN or it is otherwise recognized that the current local schedule might now be improved, the Scheduler is invoked to revise the current solution and install a new schedule. Whenever local schedule constraints change either in response to a current state update or through manipulation by the Scheduler, the DSM is invoked to communicate these changes to interested agents (i.e., those agents that share dependencies and have overlapping subjective views).

After responding locally to a given state update and communicating consequences, the agent will use any remaining computation time to explore possibilities for improvement through joint change. The Option Manager utilizes the Scheduler (in this case in hypothetical mode) to generate one or more non-local options, i.e., identifying changes to the schedule of one or more other agents that will enable the local agent to raise the quality of its schedule. These options are formulated and communicated as queries to the appropriate remote agents, who in turn hypothetically evaluate the impact of proposed changes from their...
local perspective. In those cases where global improvement is verified, the agents commit to the joint changes.

3 Reasoning about Scheduling and Coordinating

Both the Scheduler and the Options Manager compete for computational resources in the same execution thread within the CMU agent, and hence, a key control decision concerns how much time to allocate to each of these respective activities as execution proceeds. By default assumption, explicit coordination actions (issuing queries to other agents, generating options in response to queries, etc.) are given lower priority than local model updating and schedule revision actions. However, within this default structure there are still many degrees of freedom in controlling and interleaving scheduling and coordination processes. The operation of the options manager can be constrained in its frequency of use, in the type and number of options that it generates, and in the duration of the “freeze” period where options can change activities on the schedule only after that period. Likewise, the scheduler’s operation can also be constrained by limiting how frequently it is called and the amount of search performed. Ideally, the setting of parameters relevant to these different scheduler and option manager “configurations” should be driven dynamically by characteristics of the current control state.

In this section, we describe the various control parameters associated with the Scheduler and Coordinator in more detail and hypothesize desirable settings as a function of current control circumstances. Then, in Section 4, we identify a specific subset of parameters of interest and describe a series of experiments aimed at demonstrating the benefit of dynamically managing these parameter settings as a function of the agent’s current control state.

3.1 Managing the execution of the Scheduler

The scheduler component of the CMU agent is designed to incrementally maintain a high-quality local schedule as the dynamics of execution unfolds. In brief, it operates through iterative application of two subprocedures: a quality propagator and an activity allocator. Upon invocation, the quality propagator is first applied to compute the set of activities that (if scheduled) would maximize overall quality from the agent’s local viewpoint. The activity allocator then takes this set of contributors, unschedules all activities in the current schedule that are not in this set, and then attempts to incrementally insert all currently unscheduled contributors into the current schedule. If at any point during this last step, the activity allocator is unable to feasibly add an unscheduled contributor activity into the schedule, this activity is marked as “nogood” (i.e., it is “unschedulable” together with the set of activities already in the schedule). It is removed from consideration and the quality propagator is reinvoked to compute a new set of contributors, and the process continues. The scheduler terminates when the set of unscheduled contributors becomes empty (i.e., either all have been inserted
into the schedule, or some subset has been ruled out and the set of substitutable
activities has been exhausted). The reader is referred to [1] for further details.

There are a couple of basic options for controlling the amount of search
performed by the above procedure:

- **Satisfaction of soft constraints when placing unscheduled contributor activ-
  ities** (*explore-facilitated-choices*): The operation of inserting a new
  activity into the current agent schedule consists of finding a feasible “slot”
  (position) in the current scheduled sequence. In the absence of soft con-
  straints (which in this context act to boost the quality of a given activity if
  appropriate precedence relations can be established with one or more other
  *facilitator* activities), all feasible slots are equally good and hence search
  can be streamlined by simply taking the first slot found. This in fact is the
  default mode of operation, wherein satisfaction of soft constraints is only
  achieved serendipitously. However, in scenarios rich with *facilitates* possibil-
  ities, better quality solutions can be found by enumerating and selecting from
  among all feasible slots.

- **Resetting of previously established “nogoods”** (*reset-nogoods*): By default,
  an activity determined to be “nogood” during an attempt to place it on
  the timeline will remain nogood across scheduler calls. The rationale is that
  since the schedule is changing incrementally, it is likely that the set of ac-
  tivities in the schedule for which a potential contributor activity that has
  been determined to be nogood is likely to persist over time, and hence, the
  added computational expense of redetermining that these potential contrib-
  utors are nogood on each call can be avoided. However, in circumstances of
  high volatility in the local schedule, perhaps due to forced changes in inter-
  dependent decisions in other agents’ schedules or to the introduction of new
  higher priority tasks that dramatically impact the set of contributor activities,
  it can be quite suboptimal to not reopen the set of nogood activities for
  reconsideration (since they were determined heuristically in the first place).

In addition to these basic configuration options, there are a few additional
ways in which the use of the scheduler can be modulated:

- **When to reschedule?** (*reschedule-strategy*): One of the basic assumptions
  of the CMU agent design is that its underlying “flexible times” representa-
  tion of the schedule provides a basic hedge against uncertainty. Accordingly,
  the agent operates with a default policy wherein the scheduler is invoked to
  revise the schedule only when the results of execution take the agent outside
  of the set of feasible evolutions of the future delineated by the current sched-
  ule. Of course, this policy can be conservative and miss opportunities for
  optimization (e.g., when activities finish early, etc.). In circumstances where
  scheduled activities have high uncertainty, a more aggressive rescheduling
  strategy that invokes the scheduler whenever execution results deviate from
  expectations by more than an established threshold can provide a better
  option.
– *Inserting redundant, fallback options to cope with uncertainty (backfill enabled):* A second frequency of use option concerns the computation and insertion of additional backup activities that are not contributor activities in the maximal quality sense but may compensate in circumstances where primary contributor activities can fail with significant probability. In circumstances where computational load is reasonable, the use of “backfilling” scheduler runs can certainly enhance the robustness of the agent’s schedule. However, in situations where the agent’s computational load is high, use of this additional computation may work against the agent’s ability to keep pace with execution.

3.2 Managing the execution of the Options Manager

The Options Manager is designed to identify and evaluate opportunities for improving the quality of the current schedule through joint change by two or more agents. At present, improvement opportunities center around the establishment of new “enablement” chains, which, in essence, establish preconditions that allow a currently unscheduled local activity to be scheduled. When invoked, the Options manager uses the scheduler in hypothetical mode to compute the maximum quality local schedule(s) that could be achieved if various remote enabler activities were assumed to be scheduled (rather than remaining in their current unscheduled state). The output of the scheduler in this mode is a set of non-local options, each of which indicates the new expected quality and the set of enabler activities of other agents that must be scheduled for this option to be taken. From this set, the option manager determines if there is an option with the highest quality gain, and, if so, issues a query to each agent owning an enabler activity identified in this option. The query requests the maximum quality that the remote agent could attain, given that the enabler activity must be included in its new schedule. Upon receiving responses to the issued queries and determining that the non-local option does indeed boost overall quality, the options manager issues messages to all parties to commit to this option.

Similar to the scheduling process, there are several parameters for controlling this explicit coordination process:

– *Frequency of use (coordination-frequency):* One basic parameter specifies how frequently to trigger the non-local option generation process. Within the blackboard control framework, non-local option generation actions are generally given lower priority than model updating or rescheduling actions. However, there is still an issue of how frequently to attempt to generate non-local options, since once such an action is initiated it cannot be preempted. $\text{coordination-frequency} = n$ simply specifies that a non-local option generation action will be queued for execution once every $n$ ticks. In situations of high dynamics (demanding frequent updating, rescheduling and DSM communication among agents), it makes intuitive sense to dial down (decrease) $\text{coordination-frequency}$. Alternative, in situations of low dynamics, it makes sense to accelerate the search for productive multi-agent change.
“On demand” triggering delay period (quiescence period): A second manner in which the option manager and explicit coordination can be triggered is “on demand”, i.e., in response to some specific event. The receipt of a new task is a good example of this sort of event, as the integration of the local activities implied by this task into the agent’s schedule may require an enablement activity to be executed by another agent. In this case, integration of the new task, discovery of all inter-dependencies with the activities of other agents, and establishment of a new local schedule that reflects this new task may take a few ticks. To initiate non-local option generation prior to this point is not likely to be productive. The quiescence period parameter provides a knob for calibrating the timing of an “on demand” response.

Number of options generated (nbr-non-locals): The number of options that are generated on any given call to the Options Manager is a third parameter. By default, the system currently generates only a single non-local option at a time. However, in situations where dynamics are low and the computational load is relatively low, increasing the number of non-local options generated can broaden the search for productive multi-agent change.

Freeze period for generating non-local changes (non-local-freeze-period): This parameter on the non-local option generation process stipulates a time window relative to current-time within which agents involved in the non-local option are precluded from either altering existing scheduled activities or allocating new ones. Since it takes time to coordinate with other agents, the viability of negotiated scheduling changes depends on advance planning that accounts for time likely to be consumed in communicating and committing to them. Through experience with the agent, we have established a nominal default value for this non-local-freeze-period. However, in situations of high dynamics and high communication overhead, we expect that it may be advantageous to increase this horizon.

Response Priority (response-priority): When an agent receives a query from an agent pursuing an explicit coordination session, it has to determine when to allot processing time to generate the response. While the response is being generated, processing of updates and scheduling may be delayed. The default policy is to make the response-priority low, which means that, while there are update calls and scheduling calls on the control agenda, they are processed before the response is generated. While this policy allows the agent to keep up to date with critical updates, the trade-off is an increased likelihood of failed coordination sessions as query responses may be delayed such that the initiating agent cannot secure a joint commit in time for the earliest necessary activities to start. Thus, in some situations, it may be beneficial to raise the priority of the responses, so that they may be completed in a timely manner.

3.3 Control Parameters

Table 1 summarizes the meta-level control parameters available for managing the interplay between local scheduling and non-local coordination in the CMU
Control Parameter | Possible Values
--- | ---
coordination-frequency | low, high
quiescence period | low, high
nbr-non-locals | 1, ..., n
non-local-freeze-period | normal, extended
response-priority | low, high
reschedule-strategy | lazy, aggressive
explore-facilitated-choices | t, nil
reset-nogoods | t, nil
backfill-enabled | t, nil

Table 1. Control Parameters in the CMU Agent

Agent. For purposes of reasoning about the setting of these parameters, we make qualitative value assumptions. In the next section, we identify and focus on a specific subset of parameters with the goal of demonstrating the leverage to be gained by managing them dynamically over the course of a given problem solving run as a function of the evolving control state.

4 An Empirical Analysis of the Benefit of Meta-Reasoning

We focus here on the impact of the above defined meta-level control parameters that are most relevant to boosting quality via explicit coordination between agents. The sequence in which such coordination might unfold is illustrated for a simple 2-agent example in Figure 2. The 3 frames show the subjective views of the two agents and their respective schedules as they respond to the arrival of a new task structure (the T2-1 subtree) for Agent Gray. The representation here reflects the ctaems formalism of specifying quality for leaf nodes (along with other attributes such as activity duration and probability distributions not shown) and a quality-accumulation function (QAF) that specifies how the qualities of scheduled children are aggregated. Communications between agents, such as the messages from an agent’s DSM in transmitting assessed quality for a task, are depicted with dashed lines between the agent names at the top of each pane in the figure.

The snapshot in the first frame shows Agent Gray’s schedule shortly after the new T2-1 task subtree (along with the new enables constraint) is added to its subjective view. As a result of local scheduling, Agent Gray has boosted local quality by adding the NEW2 activity to its timeline. Note that it is not useful for Gray to schedule the higher quality NEW1 activity since Agent White has not scheduled the required enabler activity, M2. Agent White’s schedule is unchanged by the arrival of T2-1 since its quality propagator finds no improvements over its baseline schedule.

In the second frame, Agent Gray’s Option Manager has requested a scheduling pass for non-local options and it identifies an opportunity to boost local assessed quality if Agent White cooperates. Gray then issues a query requesting
Agent Gray finds higher quality option requiring coordination with Agent White — sends query w/ priority and freeze period.

Agent White responds with quality change induced by request.

Agent Gray assesses net gain, issues 'commit'.

New tasks arrive for Agent Gray — Scheduling adds NEW2.

---

Fig. 2. Explicit Coordination Example.
Agent White to schedule the enables source activity M2 and report back the change in its local quality assessment. (Not shown is the unrolling that each agent does after these hypothetical scheduling operations.) In the third frame, Agent White responds to the query by sending back that scheduling the enables source reduces its local quality by 5. Agent Gray determines that there is a net gain of 15, issues a commit directive to Agent White, and re-inserts NEW1 on its timeline per the option that generated the query.

4.1 Experimental Design

Ultimately, as our understanding and refinement of the explicit coordination component matures, this study will be extended to characterize the first four parameters of Table 1 which shape key aspects of explicit coordination. This will entail extensive runs over the large, randomly generated problem sets typical of the Coordinators competitive evaluations. For this scoping study we restrict our analysis to a few hand-generated scenarios that illustrate the impact of dynamically modulating one of the explicit coordination control parameters on option generation and selection. Here we isolate the impact of the control parameter non-local-freeze-period used by the initiating agent to select the best option to pursue, i.e., the one that will have both the best chance to be coordinated successfully and the most additional quality.

As defined above, non-local-freeze-period specifies the time window within which an agent scheduler is not allowed to modify its existing schedule. Intuitively, the smaller that window is, the more options that may be potentially discovered. The larger the window is, the more time that is available for an explicit coordination session to complete. Our hypothesis is that the best setting for non-local-freeze-period is context-dependent and that no single setting is best for all contexts. Our intuition is that the processing stress on an agent over a suitable recent time window (henceforth, referred to as the “process load”) will be a good predictor of the current context and, consequently, of the appropriate setting. We suspect that the heavier an agent’s problem load is, the longer the non-local-freeze-period should be, since both its own negotiation response time and that of other involved agents are likely to lag due to many competing updates.

To measure an agent’s process load, we measure the average latency of jobs on the control agenda from when they arrive until they are processed. Such jobs include updates from the simulator, updates and queries from other agents, scheduling invocations, as well as calls for generation of non-local options, which the agent imposes on itself. Specifically, this latency corresponds to the average time that a job sits on the blackboard control agenda until they are selected and processed. The longer the latency is, the busier the agent is.

For the purpose of this experiment, we have simplified the set of possible values for both the control parameter settings and the process-load measure. The non-local-freeze-period parameter can be either normal or extended, and an agent’s process load can be either low or high. The effect of an agent’s process-load on the control parameter settings is assessed according to the quality
gain accumulated at the end of the run relative to the agent running in a baseline explicit coordination configuration. This baseline configuration is described later in this section.

<table>
<thead>
<tr>
<th>non-local-freeze-period</th>
<th>response-priority</th>
<th>process load</th>
<th>quality gain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>low</td>
<td>low</td>
<td>no</td>
</tr>
<tr>
<td>normal</td>
<td>low</td>
<td>high</td>
<td>yes</td>
</tr>
<tr>
<td>extended</td>
<td>low</td>
<td>low</td>
<td>yes</td>
</tr>
<tr>
<td>extended</td>
<td>low</td>
<td>high</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. 2-Term Model of Meta-Cognition for Pursuing Explicit Coordination

A longer-term goal of this study is to learn the model for these settings using a training set of problems. Table 2 shows an example portion of such a learned model for two of the four parameters directly involved in explicit coordination: non-local-freeze-period and response-priority. However, we defer further discussion of this effort to learn the best setting to the concluding remarks in Section 5 and restrict this scoping study to a set of model values that is based on logged experience working with the agent. We focus next on how such a model is used during scenario execution.

Suppose new activities arrive that present the possibility of a non-local coordination. Each agent owning or having visibility of portions of the new task structures first enter a brief “quiescent period” during which they exchange certain information they need to effectively coordinate over the new tasks [1]. Once they exit this quiescent period, each agent adopts an anytime scheduling policy in that it first seeks improvements to its existing schedule based only on local changes it can implement and then, some time later, initiates an explicit coordination session. The behavior of the agent subsequent to the local scheduling pass is the focus here. The “baseline configuration” of the agent can be summarized as follows:

- Trigger non-local options search after 5 time ticks following the quiescent period.
- All agents use a default non-local-freeze-period of 2 ticks and a default response-priority of low.
- Sort options by quality and pursue the highest quality option (by transmitting it as one or more queries) if it improves current quality as estimated by the initiating agent.
- Commit to option if net gain is positive when all involved agents respond.

The meta-level control we propose for explicit coordination (given the arrival of new activities with opportunity for non-local coordination) consists of the following three phases:
1. Configure the non-local scheduling pass: The current value of the process load is used to set `non-local-freeze-period` for a scheduling call to find non-local options. If both normal and extended settings indicate a quality gain according to the model, then normal is used to avoid missing opportunities.

2. Select a returned option: When the scheduler returns non-local options, sort the options by quality and tag each option with the number of remote agents involved. Then choose the highest quality option with positive quality gain, breaking ties for options with the same quality by choosing the one involving fewer agents.

3. Given a feasible, quality-boosting, non-local option, configure it as one or more queries and transmit to relevant agents along with the `non-local-freeze-period`.

Each agent that receives such a query must then enforce the `non-local-freeze-period` when conducting its scheduling pass as it attempts to schedule the requested `enabler` activity(s) while respecting near-term activities and restricted space on the agent’s timeline.

The chosen meta-cognition dynamics were examined over 12 problems generated across two problems classes which can be characterized as:

1. Overall low stress conditions in which agents receive a new task with a near-term deadline that requires rapid explicit coordination to achieve maximum quality.
2. Overall high stress conditions in which too much and/or too early explicit coordination activity can potentially interfere with the more urgent need to respond to a flurry of updates coming from environmental dynamics.

The stress level on agents for these scenarios was induced by modulating the number of dynamic events impacting each agent around the approximate arrival time of new tasks. These newly arriving tasks offer the opportunity for highest quality gains if explicit coordination is successful but does not interfere with the agent’s ability to keep pace with execution. These dynamic events are reflected in the "process load" of the agents which, in turn, selects for a normal (2 ticks) or extended (5 ticks) setting of the `non-local-freeze-period` parameter.

4.2 Results

Initial experimentation to characterize process load revealed that it fluctuated rapidly within highly dynamic scenarios, in some cases backing up more than fifty updates and activities on an agent’s control agenda, but at times processing the entire agenda within a tick. As a result, low stress scenarios representative of the first class were relatively easy to generate (they could be kept small in size) and tended to produce predictable and reproducible results:

- The normal `non-local-freeze-period` value of 2 ticks provided ample time for agents involved in explicit coordination to complete negotiations and successfully accrue the associated quality boost.
All observed coordination episodes in low stress scenarios required less than 1 tick from initiation of the search for non-local options to a final commitment of agents’ schedules.

Fixing the non-local-freeze-period value to extended for these scenarios never improved performance and degraded it in terms of overall quality for three of the six scenarios.

We experienced considerably greater difficulty in definitively characterizing meta-cognition performance for the high stress scenarios. Even for low dynamics scenarios involving many agents the vagaries of execution are such that final accrued quality can be quite sensitive to such things as small communication delays across platforms/machines. As a result, identical performance is often difficult to reproduce across multiple runs of the same scenarios. This effect is compounded for 'high stress' scenarios as they depend on inducing agent congestion via injecting concentrations of many cascading dynamic changes during the scenario run. Such scenarios require much larger task structures and the impact of any one attribute such as non-local-freeze-period on overall quality accumulation can be masked by myriad other uncertainties of communication and execution. Our experience with the six high stress scenarios:

- When running under an non-local-freeze-period value of extended during the interval surrounding arrival of new tasks, 1 of 6 scenarios resulted in significantly higher quality compared with running with an non-local-freeze-period fixed at normal.
- Running with an non-local-freeze-period of normal in high stress intervals never produced higher quality than the extended setting.
- Examination of explicit coordination logs revealed that often agents burdened with over-ambitious explicit coordination attempts during high stress situations correctly addressed the most pressing updates first due to the default low priority of explicit coordination (response-priority).
- Run-to-run variations on quality of 5 - 10% were noted even with the same parameter settings.

These scoping results suggest that we may not have yet adequately simulated a sufficiently high dynamic stress level to observe the impact, if any, of an extended setting of non-local-freeze-period. Moreover, the problem sets need to be expanded considerably in order to determine whether there is a statistical benefit to the extension of non-local-freeze-period for these high stress situations.

5 Concluding Remarks

This paper focuses on the benefit of dynamically switching configurations of an agent as the execution environment changes. In the presented experiment, we engineer a control model for the configuration switching based on our observation of the agents’ performances over a large number of problems. While engineering the model is feasible in specific, easily identifiable situations, it does not represent a general solution. The complexities of the interactions among the control
parameters and the features of the environment make it difficult to generate models for a number of control parameters operating in a varied environment. For a broader solution, we would like to have agents learn control models for the appropriate actions for different contexts by classifying the performance of different configurations of the agent over those contexts.

To support this learning, we are investigating using MetaMod, a meta-reasoning component based on [5]. Using a classification algorithm, such as the naive bayes classifier algorithm provided by MetaMod, the agent will learn the model by being trained over a set of problems, where each contains specific events that exercise the control parameters being trained and the overall set presents a variety of values for the selected environment features. All problems will be run with all combinations of the control parameter settings, while measuring selected features of the environment and of the agent’s state. Once learned, the model will be applied in the same as manner as is described above in Section 4.

6 Acknowledgements

This paper is based on work supported by the Department of Defense Advance Research Projects Agency (DARPA) under Contract # FA8750-05-C-0033. Any opinions findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of DARPA.

References