Appears in Proc. Third International Workshop on Self Adaptive Software
Rosslyn, VA, June 2003

Deliberation Scheduling Strategies for Adaptive Mission Planning in Real-Time Environments

David J. Musliner
Honeywell Laboratories
musliner@htc.honeywell.com

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.info

Kurt D. Krebsbach
Lawrence University
kurt.krebsbach@lawrence.edu

1 Introduction

We are developing the Multi-Agent Self-Adaptive Cooperative Intelligent Real-Time Control Architecture (MASA-CIRCA) to address high-stakes, mission-critical, hazardous domains. [5, 6]. MASA-CIRCA is a domain-independent architecture for intelligent, self-adaptive autonomous control systems that can be applied to hard real-time, mission-critical applications. MASA-CIRCA includes a Controller Synthesis Module (CSM) that can automatically synthesize reactive controllers for environments that include timed discrete dynamics. In order to best tailor its behavior to the current context, MASA-CIRCA will sequence through a number of different controllers, one for each phase of the mission (see Figure 1).

This controller synthesis process can occur both offline, before the system begins operating in the environment, and online, during execution of phase controllers. Online controller synthesis is used to adapt to changing circumstances and to continually improve the quality of controllers for current and future mission phases. The controller synthesis process operates under the control of MASA-CIRCA's Adaptive Mission Planner (AMP). In general, the MASA-CIRCA agent is overconstrained in the sense that it cannot produce optimal plans for all phases in time to be of use. The issue is particularly acute because the controller synthesis problem is intractable in the worst case.

Because of this *bounded rationality*, the AMP must actively manage the CSM's inference. There is no point in developing an optimal controller if that controller is fielded long after it is no longer needed. This inference management problem is the problem of *deliberation scheduling*, and has been the center of much of our recent work on MASA-CIRCA. We take an approximate decision-theoretic approach to the MASA-CIRCA deliberation scheduling problem: decision-theoretic, because we attempt to optimally allocate the CSM's reasoning time; approximate because full formulations of the problem are intractable and some formulations involve an infinite regress.

We are exploring the issues of real-time intelligent control in the context of managing the autonomous controls of a self-adaptive unmanned aerial vehicle (UAV). The adaptation may be necessary for a variety of reasons – because the mission is changed in-flight, because some aircraft equipment fails or is damaged, because the weather does not cooperate, or perhaps because its original mission plans were formed quickly and were never optimized. For this reason, this controller synthesis can occur both offline, before the system begins operating in the environment, and

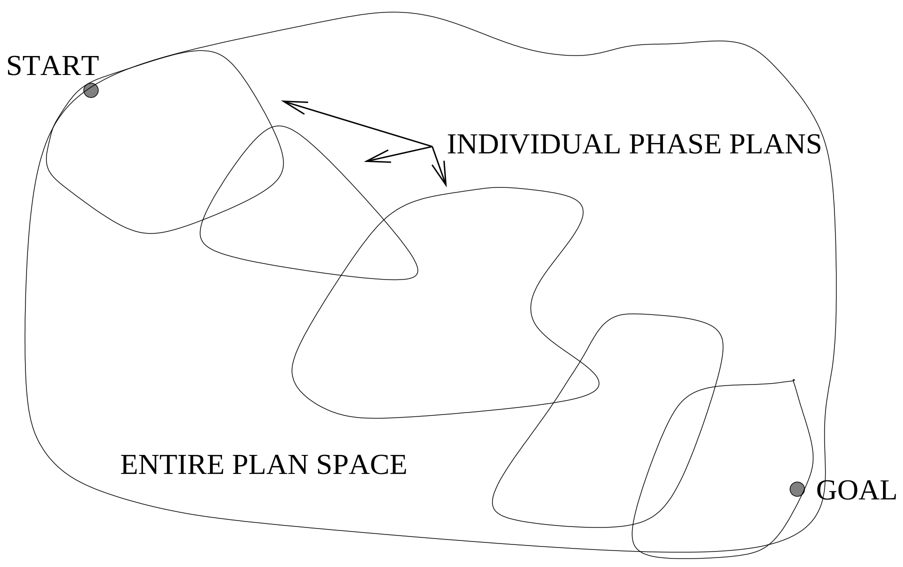

Figure 1: MASA-CIRCA agents sequence through multiple timed controllers over the course of multiple plan phases.

online, during execution of phase plans. Online controller synthesis is used to adapt to changing circumstances and to continually improve the quality of controllers for current and future mission phases.

In previous papers, we have framed the problem of deliberation scheduling for MASA-CIRCA. We described how MASA-CIRCA can modify problems it is trying to solve to adjust planning time by trading off plan quality [3]. In a later paper [1], we described how to manage the tradeoff process, i.e., how to decide what part of a mission control plan we should attempt to improve at each point of time. We presented a Markov Decision Process (MDP) model of the deliberation scheduling problem. Since the MDP is very difficult to solve, we also presented greedy (myopic) approximations to the optimal solution. In those experiments we showed that a discounted myopic approximation technique provided good performance with very limited computational costs. We also compared the performance of the discounted greedy approximation with other strawman agents that attempt to manage deliberation using easy-to-compute heuristics.

In this paper we describe how we have integrated deliberation scheduling into the MASA-CIRCA AMP. We present results on the performance of several agents in an example scenario from the UAV domain. Once again, we compare the greedy approximation technique with other strawman agents. This paper shows how the qualitatively different behaviors of different deliberation managers affects mission performance.

2 Background

2.1 The MASA-CIRCA Architecture

We work on deliberation scheduling in the context of CIRCA, the Cooperative Intelligent Real-Time Control Architecture. As illustrated in Figure 2, CIRCA combines on-line AI planning and formal verification methods that generate customized plans (controllers) with real-time reactive

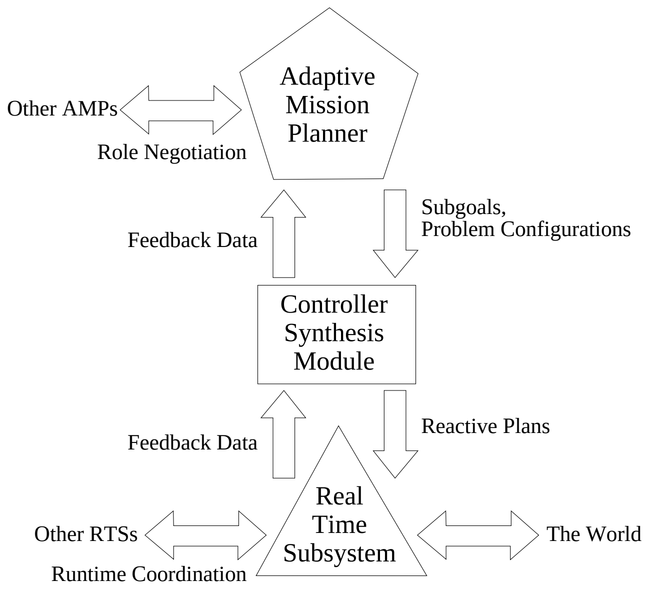

Figure 2: The CIRCA architecture.

execution of those plans. CIRCA was designed to control autonomous systems operating in dynamic and hazardous domains where real-time performance is crucial. For example, when controlling an Unmanned Aerial Vehicle the system may face a wide variety of hazardous situations (e.g., enemy threats such as surface-to-air missiles, equipment failures) that require responses before environmentally-imposed deadlines (e.g., the arrival of an incoming missile). Furthermore, the full spectrum of these threats may exceed the system's ability to monitor and respond, so that it must focus its attention on only the most likely or most hazardous in different situations. In that case, the system must provide mission-specific control plans (linking active sensing to responses) that may change as the mission progresses.

For example, a UAV may have to focus its threat detection and sensor processing hardware on different tasks throughout a mission, shifting from defensive threat detection during an ingress to surveillance and target recognition during the reconnaissance or attack phase. While pre-mission planning may be used to create customized controllers for the anticipated mission profile, on-the-fly planning may also be required when the mission changes or unexpected situations arise (e.g., unexpected enemy activity or equipment failures).

CIRCA's Controller Synthesis Module (CSM) is designed to construct such customized control plans using a combination of heuristic state space search and formal verification []. The CSM takes in a problem description that specifies the anticipated initial situation, goals (desired state characteristics), threats (uncontrollable environmental transitions) and possible actions (controllable transitions). The transition models include timing characteristics that specify how fast the various transitions can occur. Using this information, the CSM searches for an assignment of controllable actions to each anticipated reachable state, such that the resulting state space is closed and drives the system towards its goals while avoiding specified forms of failure. The CSM uses formal verification techniques to check its plans and ensure that failures are not reachable [2].

For the purposes of this paper, the key point here is that the CSM is a very complex planning system whose runtime is highly variable, since it solves several intractable problems in the process of generating new plans. Thus building new plans on-the-fly (or even during limited pre-mission planning time) can be a challenging task, and requires active control by the higher layer of CIRCA, the Adaptive Mission Planner (AMP).

The CIRCA Adaptive Mission Planner (AMP) is responsible for the highest-level control of a CIRCA agent [5, 6], determining and modifying the agent's responsibilities (threats to handle, mission goals to achieve), controlling the agent's reasoning (what plans to construct), and managing the agent's deliberation resources (i.e., how best to use computation time to improve the overall mission plan). More specifically, the AMP manages the agent's responsibilities by negotiating with other agents via contract bidding. It controls the agent's reasoning both by modifying problem configurations for the CSM, and by invoking (or halting) the CSM when appropriate. Finally, the AMP manages the agent's deliberation resources by scheduling the CSM to improve certain plans in a manner that yields the highest utility for the mission plan as a whole.

2.2 Problem Structure

A team of MASA-CIRCA agents will be given missions that are divided up into *phases*. The phases correspond to modes or time intervals that share a set of common goals, threats, and dynamics. For example, our military UAV scenarios include missions that have phases such as ingress, attack, and egress. The ingress phase is distinguished from the attack phase both by the characteristics of the flight path (e.g., a nap-of-earth stealthy approach vs. a popup maneuver very near a target) and by the expected threats (e.g., the types of missile threats present at different altitudes) and goals (e.g., reaching the target zone vs. deploying a weapon).

The team must arrange to have agents take responsibility for different goals and threats, depending on their available capabilities and resources (e.g., ECM equipment and weapons loadout). These goals and threats vary from one phase to the next. In fact, the mission is typically split into phases specifically to decompose the overall mission into manageable chunks aligned with a common set of threats, or a common goal which, when achieved, signals the end of that phase. The team of agents will allocate tasks among itself by a bidding process, with agents having different endowments depending on how appropriate they are to handle a given task.

For each mission phase, each MASA-CIRCA agent must have a plan (or controller) that is custom-designed to make the best possible effort to achieve the goals and defeat the threats associated with the phase. These controllers may be generated before or during mission execution, depending on the circumstances. The Controller Synthesis Module (CSM), described elsewhere, is capable of automatically building these controllers, but this controller synthesis can be a complex and time-consuming process.

The complexity (and hence duration) of the synthesis process can be controlled by varying the *problem configuration* that is passed to the CSM. The problem configuration describes the characteristics of the desired controller for a particular mission phase [4]. The problem configuration contains information about the initial state of the world, goals to achieve, threats that are present, state transitions that can happen due to the world, and actions available to the agent to affect the world. By varying these details, the AMP can make a planning problem fall anywhere in a complexity spectrum from very simple to infeasible. For example, consider an agent that must fly over a particular segment of airspace in which we expect there to be two surface to air missile (SAM) sites. We might give the CSM a problem configuration with the goal of reaching the waypoint at the end of the flight segment, and the two goals of suppressing the two SAM sites. Alternatively, if that was too difficult, we could instruct the CSM to reach the final waypoint while simply using passive countermeasures to evade the SAMs.

2.3 Predictive Deliberation Management

One of the primary responsibilities of the AMP is to determine which mission phase the CSM is trying to build a controller for at any moment, and how hard it should work to do so, by modifying the phase problem configurations. This is what we mean by the AMP's deliberation management function. In each phase, the more threats that can be handled, and the more goals that can be achieved, the higher the probability that the team will achieve its goals and survive the mission. Thus, the problem can be cast as follows: Given a set of planning phases, quality measures of the current plan for each phase, a set of tradeoff methods (i.e., improvement operators) applicable in each phase, and some amount of time to try to improve one or more of the current plans, how should the AMP allocate the next segment of time to improve the overall expected utility of the mission plan as a whole? Note that while we discuss these two aspects of the problem separately, the two aspects interact, and the decisions are made together.

To effectively decide what deliberation should happen now, the AMP must consider the potential deliberation schedule into the future. For example, the AMP might consider starting a lower-priority CSM task earlier if there is a shorter window of opportunity in which to execute that task, or the expected execution time is longer. In this case, the AMP would also need to consider whether it expects that this will still leave time to execute the higher-priority task later. As we will see, more efficient, but incomplete approaches to the problem can suffer from local maxima, and miss solutions that require this type of further lookahead and more complex analysis.

The second part of the problem is to select what improvement to apply to the selected phase. which of several improvement operators to apply to the phase it has selected. The AMP may improve a phase's controller by replanning to generate a new controller that handles more threats or more goals. Through experiments, we have developed a model of the performance of the CSM when given varying number of threats and goals to handle. For various numbers of threats and goals, we have developed cumulative probability functions that record the probability of successfully completing a CSM run as time increases. We use this to determine the payoff the agent will receive for committing time to various CSM actions. Note that since MASA-CIRCA is a hard real-time system, the CSM cannot simply add more threat- and goal-handling to improve the plan. MASA-CIRCA's RTS ability to react to multiple events is limited by the available processing power, and all reactions must be guaranteed to run to completion within their deadlines. So it is quite possible that the system cannot handle all threats and goals, and must sacrifice some to cover others.

2.4 Combinational Configurations

As discussed earlier, the AMP is responsible for downloading one problem configuration at a time for the CSM to work on. This configuration embodies a planning problem, i.e., an initial state, threats, goals, actions at the agent's disposal, etc. A *combinational configuration*, then is a problem configuration that the AMP automatically generates when it learns what threats and goals are in each phase.

The AMP will enumerate and consider combinations of different goals and threats. For instance, a stand-alone agent faced with a phase with threats T1 and T2 and goal G1, will consider synthesis for the combinations G1, T1, T2, G1+T1, G1+T2, T1+T2, and G1+T1+T2. When considering improvements to an existing plan, of course, the AMP should never consider plans that handle fewer features than the existing one. E.g. the AMP will not consider the singleton set T1 if the system already has a controller that handles T1+G1.

In a multi-agent team context, deliberation management interacts with contracting. Consider a case where the agent described above was part of a team and had won contracts to handle T1, T2, and G1. In that case, if the AMP were later to find that it could only handle T1+T2 or T1+G1, it should *renege* on its contract for either G1 or T2. In that case the contract would again become available for bids and another agent would take it over and handle it.

3 Beyond the MDP Model

In previous work, we studied a Markov Decision Process (MDP) model for MASA-CIRCA deliberation scheduling [1]. This model posed the problem of deliberation scheduling as one of choosing, for each point in time, a plan improvement operator to apply to the set of phase plans. The MDP made a number of simplifying assumptions, and was not actually integrated the MASA-CIRCA architecture. However, the MDP model provided an opportunity to evaluate a number of computationally inexpensive approximation methods for solving the deliberation scheduling problem. Because the MDP was simple, we were able to evaluate those approximation methods against the "gold standard" offered by dynamic programming algorithms for MDPs. Our early experiments indicated that a myopic (greedy) approximation to the optimal solution provided a good tradeoff of quality of results against computation time. We further showed that applying time-discounting to the myopic algorithm helped avoid some suboptimal choices in the test domains.

While the MDP model provides a useful experimentation platform for comparing deliberation scheduling algorithms in moderately complex domains, it omits several key characteristics of the problem faced by CIRCA's AMP. For example, while the abstract MDP model did represent plan modification operators (calls to the CSM) that were expected take more than one time quantum, the operators were modeled as always using a fixed amount of time. In contrast, the real CSM may return earlier than expected if it finds a plan or determines that it cannot find one. Also, the MDP model did not represent the goal-achieving quality of the plans: the MDP model concerned itself only with threat-handling and assumed that the agent would achieve goals as long as it wasn't destroyed. No plan improvement operators could increase or decrease the agent's probability of goal achievement, except indirectly by affecting its survival probability. In contrast, the real agent can build plans that are better or worse at goal achievement, and may need to tradeoff goal achievement against survival. The agents may even choose to ignore some goals; thus the real agent may survive a phase where a goal is present and yet acquire no reward.

While we could have added these characteristics to the MDP model, this would have increased the complexity of optimal solving to the point where meaningful experimentation would have been prohibitively expensive. Furthermore, the MDP model was only intended as a preliminary investigation into low-cost deliberation scheduling strategies that could operate effectively in our more challenging, more dynamic simulated UAV environment.

```
(while (not *halt*)
    (setf task (rank-and-choose #'priority #'max (tasks *self*)))
    (cond (task ;; if there is a task selected, remove and execute it.
                (setf (tasks *self*) (delete task (tasks *self*)))
                (execute-task task)
                (process-all-msgs))
          (T    ;; no tasks are ready; wait on inputs and flash heartbeat
                (if (wait-for-input-available *sockets*
                                              :timeout *heartbeat-period*)
                    (process-all-msgs)
                    (show-heartbeat)))))
```

Figure 3: Simplified Lisp code for the AMP outer loop, processing tasks and messages.

Thus we moved into building the myopic strategies investigated in the MDP model into the real AMP code, to support experimentation and evaluation with the real CSM and real online challenges. In the following section, we describe the AMP's design features that support deliberation scheduling and the extensions made to the strategies already discussed.

4 Deliberation Scheduling in the Adaptive Mission Planner

Our AMP prototype executes a fairly simple outer loop based on a "task" metaphor. Every major function that the AMP can perform is encapsulated in a task object. For example, one of the main tasks the AMP manages is telling the CSM what controller synthesis problems to work on. Controller synthesis problems are represented by "problem configuration" objects that contain all of the CSM API calls to describe the problem to the CSM. For each problem configuration that has not yet been solved by the CSM, the AMP maintains a task object which, if executed, will send the API commands to the CSM and wait for a solution in return. Similarly, the functions to support inter-agent negotiation are encapsulated in task objects. When the CSM produces an executable plan (controller) in response to a particular problem configuration, a new task is created to indicate that the new plan can be downloaded to the executive (RTS).

Tasks have associated priorities that are used to order their execution. On each cycle through the AMP outer loop, one of the highest-priority tasks is selected for execution and all waiting incoming messages are processed. If no tasks are waiting to execute, the AMP blocks (periodically flashing a heartbeat signal) until it gets an incoming message, which will trigger formation of a new task. Figure 3 shows a slightly simplified version of the Lisp code used to implement this loop.

Task priorities can be static or computed dynamically, with different computation methods depending on the class of the task object. For example, the class of tasks for downloading new plans to the RTS have static priorities set quite high, so that the AMP will almost always prioritize downloading new plans.

To implement deliberation scheduling that determines which problem configuration task should be addressed next by the CSM, the planning tasks can be configured to use a dynamic priority function that depends on many different factors. For example, we can implement deliberation

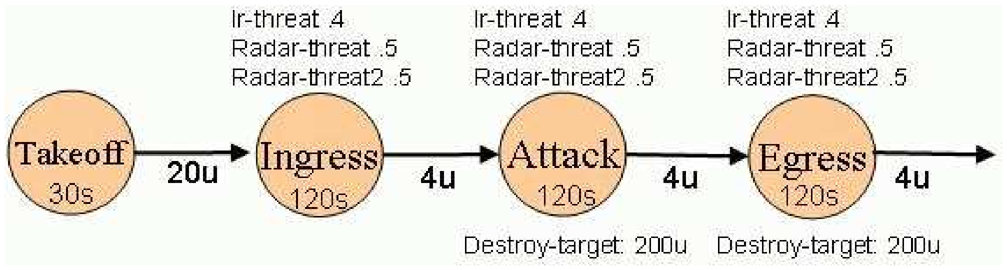

Figure 4: Each phase of the mission involves different threats and goals.

scheduling based on expected utility by having the system automatically incorporate information about the expected time to finish the task, the expected benefits of the task (e.g., the improvement in expected controller quality [and hence mission success] that will result if the CSM builds a new controller for a particular phase), and the time at which the task solution is required (e.g., when the aircraft will enter the mission phase for which this controller is intended).

5 Performance Results

5.1 Overview

In this section we describe experimental results that illustrate how the different deliberation scheduling algorithms implemented in the AMP can result in widely varying performance. In this experiment, we fly three simulated aircraft controlled by MASA-CIRCA agents through the same mission scenario. The three aircraft fly essentially the same route through an environment containing several threats and goals[1]. The three agents differ only in their use of different deliberation scheduling algorithms; the point of the scenario is to show that more intelligent deliberation scheduling algorithms can lead to dramatically improved performance results.

Figure 4 provides an overview of the mission, illustrating the sets of threats and goals present in each phase. The mission begins with a simple takeoff phase in which the aircraft face no threats, and only have the goal to reach the ingress phase, which is valued at 20 units of reward (utils). From then on, progressing to the subsequent phases earns the aircraft 4 utils on each phase transition. An aircraft may fail to progress to the next phase either by being destroyed by a missile threat or by not flying along its planned path (e.g., continuously performing evasive maneuvers).

The phases have different expected durations, corresponding to how long the aircraft take to fly each leg of the flight plan. In the figure, the expected durations are shown in seconds within the phase circle. To make this a compact demonstration, we have made some of these phases shorter than real combat missions (e.g., ingress might typically take more than two minutes).

In three of the phases, the aircraft are expected to face three kinds of missile threats, with varying degrees of hazard. For example, the IR-missile threat is expected to be 40% lethal, meaning that if the aircraft does not build a plan that anticipates and reacts to this threat, then the threat will destroy the aircraft 40% of the time it tries to execute this mission. Fortunately, this environment is considerably more lethal than real-world combat flying.

In the attack and egress phases, the aircraft also have goals to destroy two targets, valued at 200

[1]Their planned paths are identical, but their actual flown routes may differ due to variations in the use of evasive maneuvers.

utils each. If the CSM is able to build a plan that includes the actions to destroy the target, then the aircraft will accrue this value if it survives long enough to execute the associated fire-missile reactions. Building a plan that only destroys the targets is quite easy. However, building a plan that both defends against the different threats *and* destroys the targets is more time consuming. Building a plan that handles all the threats and goals is not feasible in the available mission time. As a result, the AMP must carefully control which planning problem it works on at any time.

Note that Figure 4 describes the scenario as it is described to the MASA-CIRCA agents for their planning purposes. This description includes goals that are definitely present, and threats that *may* be encountered. In the actual scenario that we flew the simulated aircraft against, the aircraft do not encounter all of the potential threats. They actually only encounter `radar-threat2` type threats in both the ingress and attack phases.

Also, to apply time pressure to the planning process, the CIRCA agents are told they must begin flying the mission as soon as they have a baseline (simple) plan for the first phase, Takeoff. Since building the takeoff plan takes well under one second, they essentially begin executing the mission as soon as the mission description arrives. All of the planning for later phases is performed literally "on the fly."

The simplest aircraft, Agent S, uses a "shortest problem first" algorithm to decide which planning problem to work on next. A more complex aircraft, Agent U, uses a greedy deliberation scheduling algorithm without discounting (highest incremental marginal utility first). The most intelligent aircraft, Agent DU, uses a discounted greedy approach (highest discounted incremental marginal utility first). The demonstration shows how the more intelligent deliberation scheduling algorithms allow the latter aircraft to accomplish more of their goals and defeat more of their threats, thus maximizing mission utility for the entire team.

5.2 Analysis of Agent S

Agent S sorts all of its possible deliberation tasks based on the expected amount of time to complete each task, preferring the shortest. Recall that a deliberation task is a request to the CSM to plan for a new problem configuration. In general, the more complex the configuration is (i.e., the more goals and threats), the longer the expected planning time.

Figure 5 illustrates which threats and goals are handled by the mission plans that Agent S developed over the course of the entire team mission. Along the vertical axis, the rows correspond to the various threats and goals in each mission phase. Time in flight is shown by the horizontal axis. Dark bars in each row indicate the time period during which the aircraft has a plan that handles the row's respective threat or goal. As the CSM completes its reasoning for a particular problem configuration for a particular mission phase, as selected by the deliberation scheduling algorithm, the new plan is downloaded to the RTS and the configuration of dark bars changes.

For example, the first row indicates that Agent S immediately constructs a plan to handle the IR-threat for the ingress phase, but quickly supplants that current plan with a plan to handle the ingress radar threat instead, as shown on the second line. In an ideal situation, when each mission phase begins the aircraft would already have a plan covering all threats and goals in the phase, and that plan would persist unchanged throughout the phase. Charted as in Figure 5, this would look something like a white staircase descending to the right, with dark bars above.

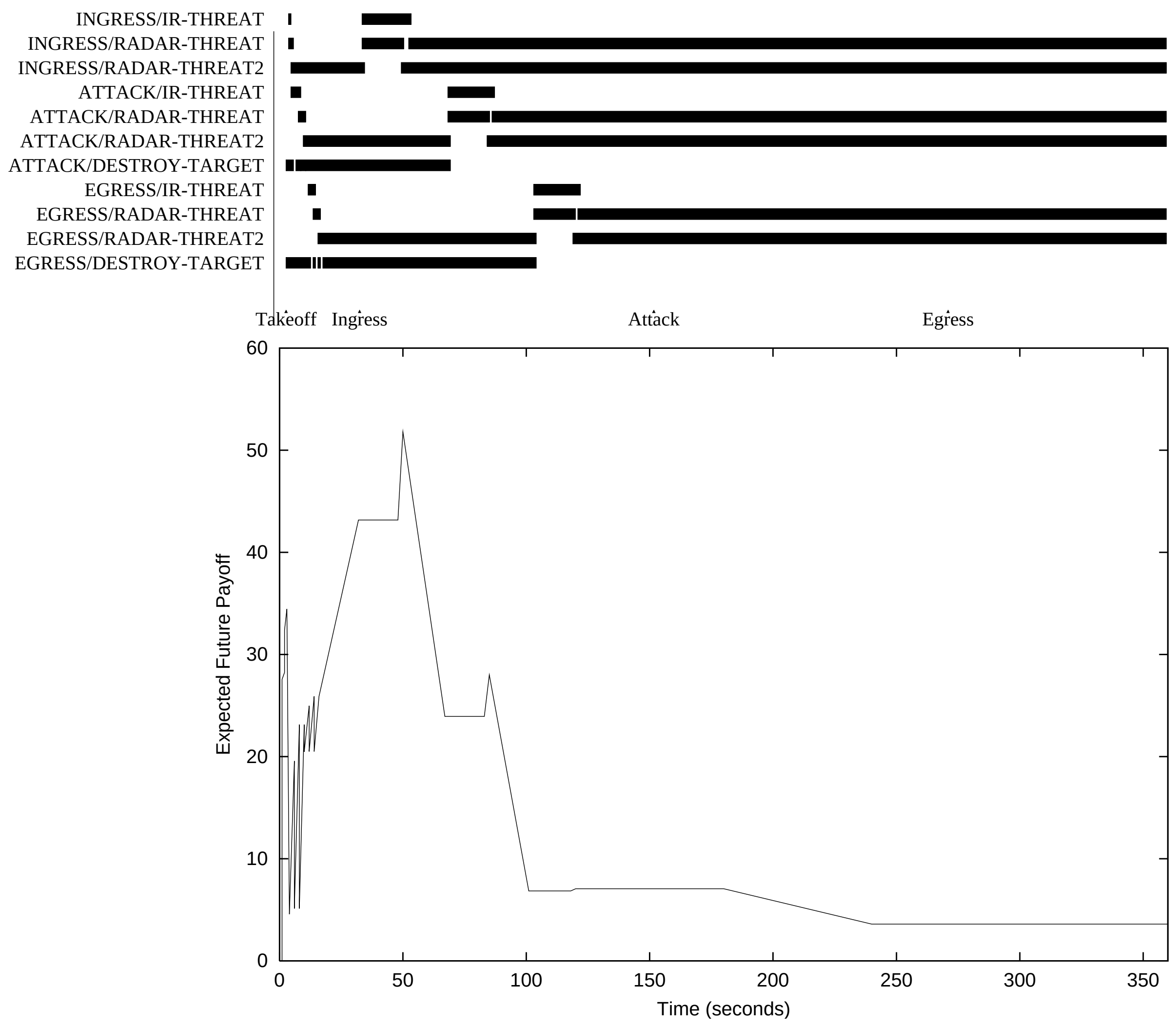

Figure 5: Gantt chart of threat and goal coverage for Agent S throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent does not have plans to destroy the targets during the appropriate phases, and thus acquires few utils.

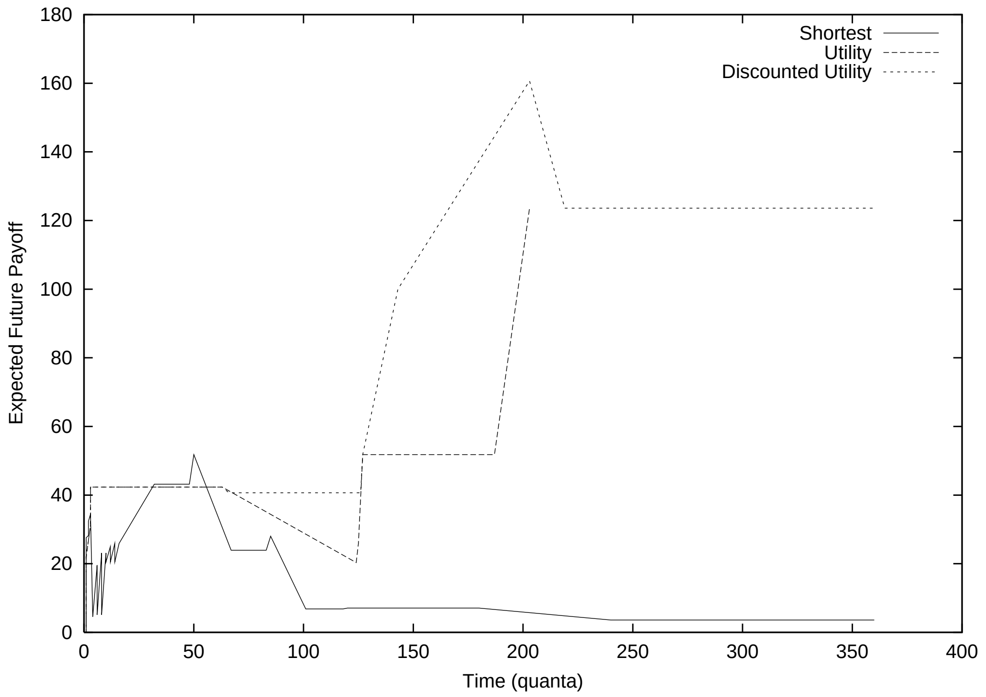

Figure 6: Each agent's expected payoff over the course of the mission.

A key characteristic of Agent S's performance is that, lacking any better way to compare two plans, the agent uses its preference for shorter-planning-time as an estimate of plan utility or quality. As a result, whenever the CSM returns a successful plan, Agent S assumes that it is better than any previously-generated plan for that mission phase and discards old plans, installing the new one. In fact, all of the agents use this same behavior, since the deliberation scheduling algorithm is expected to select for planning only those problem configurations that may lead to an improvement in overall mission performance.

However, this leads to rather erratic behavior for Agent S, mostly because there can be ties in estimated planning time for configurations with the same number of threats or goals. For example, as can be seen in all three phases for Agent S in Figure 5, Agent S covers two threats (or one threat and one goal) for a length of time, then switches to cover two others later in the mission. While the other agents may similarly change which threats and goals they handle, they do so based on expected utility measures only. Agent S's impoverished estimate of plan utility (just expected planning time) causes it to waste time generating plans that are not necessarily better than those it already has.

Figure 6 illustrates the expected future utility that each agent has as the mission progresses. This chart is somewhat complicated by the fact that it only includes *future* utility, so that as the aircraft completes phases and earns utils, its expected future utility actually drops[2]. However, comparison between the agents on this chart still shows what we hoped: Agent U and Agent DU significantly outperform Agent S in both expected future utility and, as described later, in acquired utility (corresponding to actual mission performance).

[2]The authors are currently planning to fix this problem by recording and plotting both earned and expected utility.

As shown in Figure 6, Agent S's strategy allows it to successfully defend itself against the radar-guided missile (`radar-threat2`) that may attack it in the ingress phase. When another radar missile (also `radar-threat2`) attacks it in the attack phase, it is also prepared to defeat it, as its current plan handles both `radar-threat` and `radar-threat2` threats. However, it does not handle the goal of destroying the target, and thus loses significant potential reward by not achieving the main mission goal. This failure to achieve full mission success is largely due to the fact that Agent S's heuristic is not utility-based, and thus does not distinguish between reward and survival, sometimes replacing valid, more valuable goal-achieving plans (e.g., `radar-threat2` + `destroy-target`), with non-goal-achieving plans that it considered slightly more complex based on its cost-estimation function.

5.3 Analysis of Agent Agent U

Rather than computing a policy that indicates what actions should be taken in any possible future state to maximize expected utility, Agent U myopically looks one state ahead along all of its immediate action choices and selects the action that results in the mission plan with the highest expected utility. Agent U need not compute a complete policy; instead, it computes the policy lazily, determining the action choice for each state only when queried.

Because the greedy agent is making decisions with limited lookahead, it has trouble assessing the relative merit of addressing near-term vs. far-term risks.

The threat and goal coverage history for Agent U is shown in Figure 7. Like Agent S, Agent U is able to successfully defend itself against the radar-guided missile (`radar-threat2`) attacking it in the ingress phase. However, when another radar missile attacks it in the attack phase, it is not prepared to defeat it, and it is killed. It has not done the planning for `radar-threat2` in the attack phase because it is making decisions with limited lookahead, and it has trouble assessing the relative merit of addressing near-term vs. far-term risks. This problem is exactly what discounting is meant to address. Although there was ample time available to plan for both the attack-phase radar threat and the egress-phase destroy-target goal, Agent U is "distracted" by the possibility of destroying a high-value target in the next mission phase, and busily tries to build high-quality plans for that future phase instead of for the current attack phase. As a result, it fails to develop a good defensive plan, and is destroyed by a radar-guided missile[3].

5.4 Analysis of Agent Agent DU

Finally, Figure 8 illustrates the threat and goal coverage profile for Agent DU.

Agent DU builds plans that handles all of the threats that actually occur, and achieves its goals, achieving the maximum possible mission utility. It does not make the simple mistakes that the uninformed Agent S does, because its deliberation scheduling strategy correctly trades off threat-handling and goal-achievement by computing incremental marginal utility. In addition, its discounting of later utility encourages it to defer planning for which it has more time, helping it to avoid making the greedy mistakes that Agent U is prone to.

[3]The authors are working to develop a graphical method to display what phase the agent was attempting to plan for at each time, to clarify this focus-of-attention point.

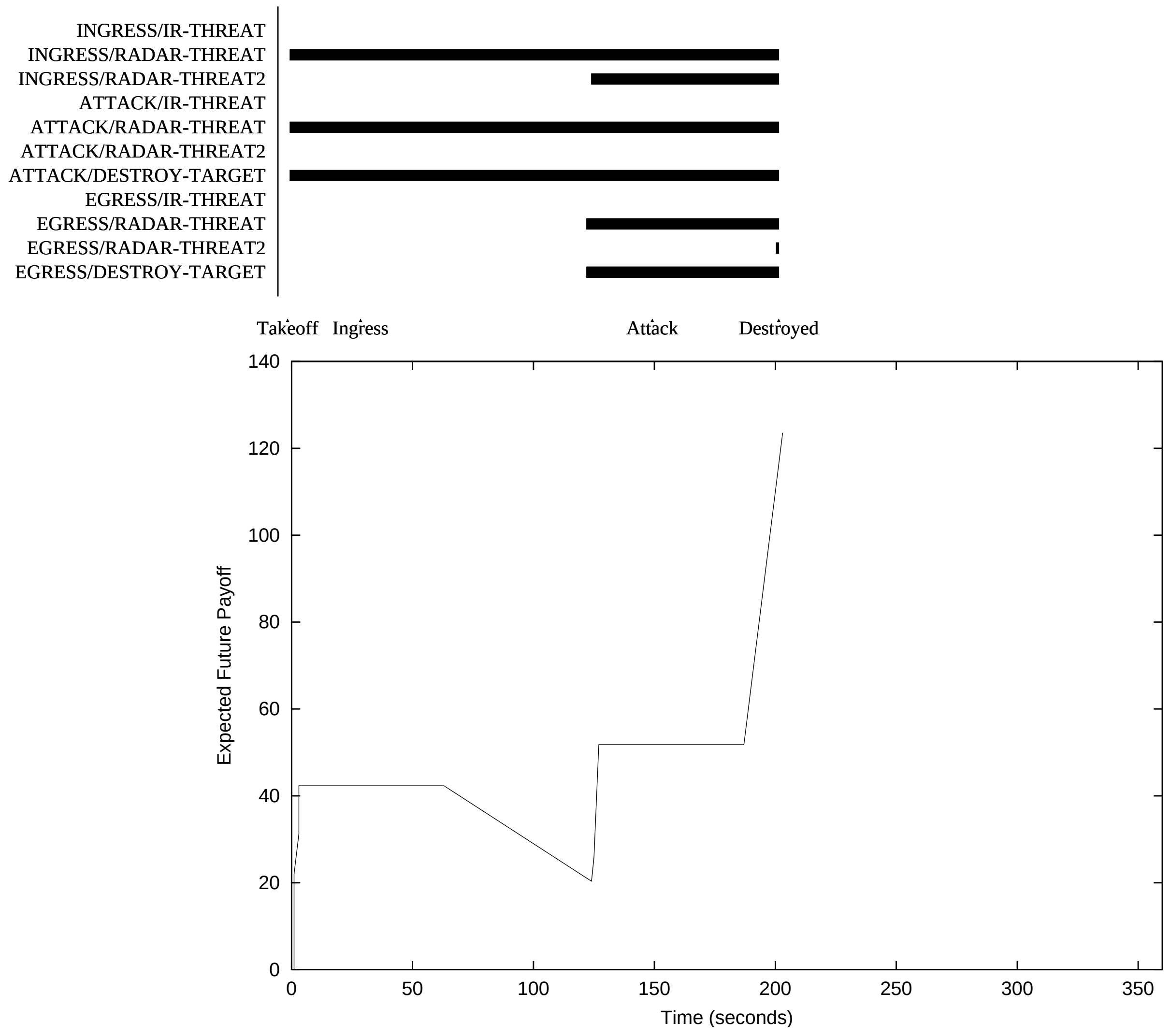

Figure 7: Gantt chart of threat and goal coverage for Agent U throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent is not prepared to defeat `radar-threat2` in the attack phase, and it is destroyed.

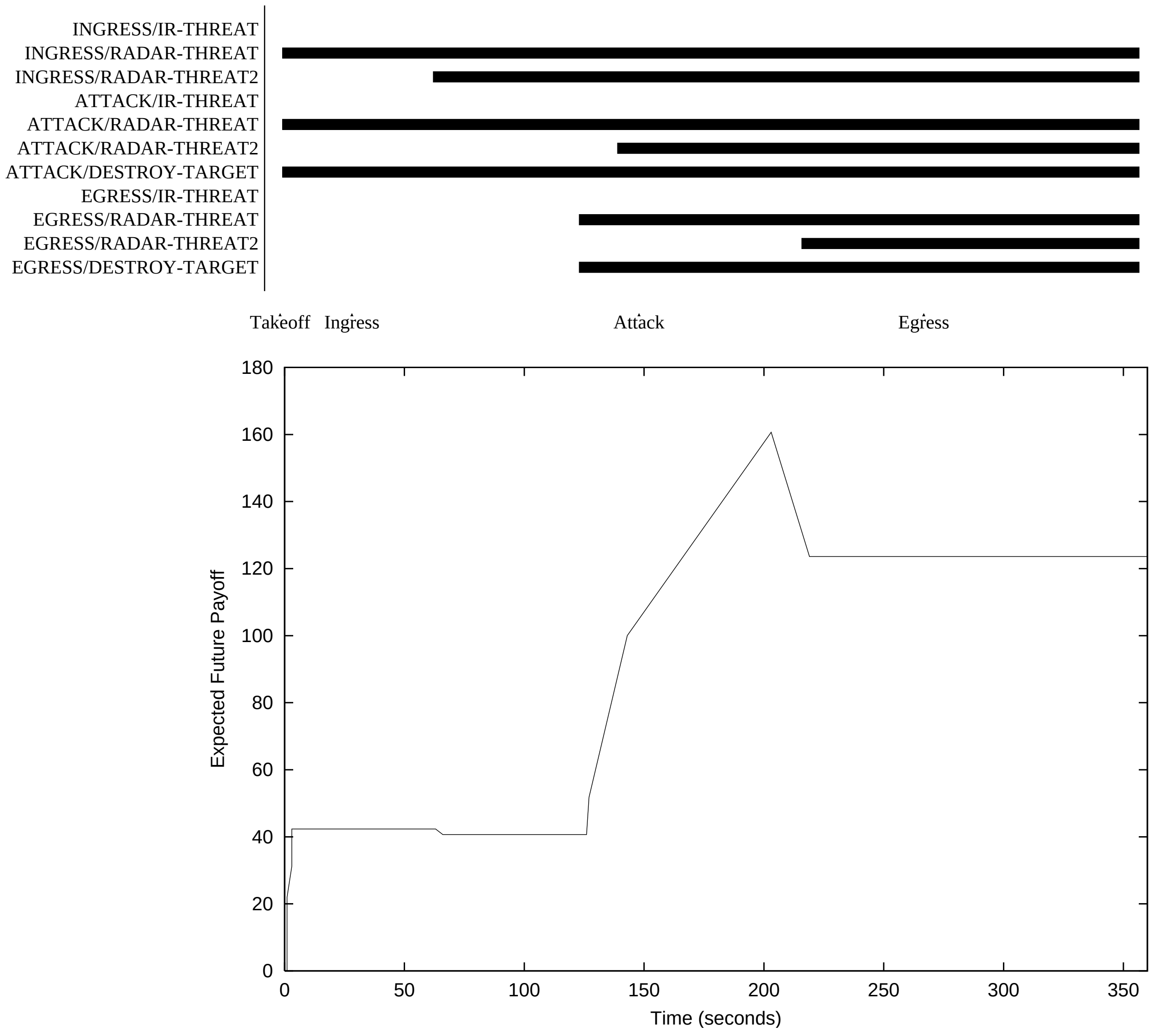

Figure 8: Gantt chart of threat and goal coverage for Agent *DU* throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent has plans to accomplish the `destroy-target` goals by the time the respective phases occur.

6 Conclusion

As described above, a preliminary version of the new AMP design incorporating several forms of deliberation scheduling is now operational. The system can be easily configured to use different deliberation scheduling algorithms by assigning different priority computation methods to classes of tasks. The demonstration scenario we described shows how the more intelligent deliberation scheduling algorithms allow Agent *DU* to accomplish more of its goals and survive longer.

One key future direction revolves around integrating our approaches to deliberation scheduling and multi-agent negotiated coordination. As noted above, over-constrained agents in teams should be able to move smoothly between tradeoff strategies in which they use on-line negotiation to re-allocate roles and responsibilities (threats and goles) or locally choose to build partial, incomplete plans that do not satisfy all of their responsibilities. We plan to use our unified deliberation scheduling scheme to allow this transparent integration of negotiation and local problem solving. Negotiation will be just one more optional task that the AMP can choose among when it is considering what deliberation task is most appropriate next. This will require performance profiles describing the expected performance of negotiation strategies, as well as time-bounded negotiation behavior.

Acknowledgments

This material is based upon work supported by DARPA/ITO and the Air Force Research Laboratory under Contract No. F30602-00-C-0017.

References

[1] R. P. Goldman, D. J. Musliner, and K. D. Krebsbach, "Managing Online Self-Adaptation in Real-Time Environments," in *Proc. Second International Workshop on Self Adaptive Software*, 2001.

[2] R. P. Goldman, D. J. Musliner, and M. J. Pelican, "Exploiting Implicit Representations in Timed Automaton Verification for Controller Synthesis," in *Hybrid Systems: Computation and Control (HSCC 2002)*, C. J. Tomlin and M. R. Greenstreet, editors, number 2289 in LNCS, pp. 225–238, March 2002.

[3] D. J. Musliner, "Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis," in *Proc. Int'l Workshop on Self-Adaptive Software*, 2000.

[4] D. J. Musliner, "Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis," in *Self-Adaptive Software*, number 1936 in LNCS, 2001.

[5] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, March 1995.

[6] D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach, "Self-Adaptive Software for Hard Real-Time Environments," *IEEE Intelligent Systems*, vol. 14, no. 4, pp. 23–29, July/August 1999.