AFRL-IF-RS-TR-2004-32
Final Technical Report
February 2004

MASA-CIRCA: MULTI-AGENT SELF-ADAPTIVE CONTROL FOR MISSION-CRITICAL SYSTEMS

Honeywell Technology Center

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J124

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-32 has been reviewed and is approved for publication.

APPROVED: /s/
DANIEL E. DASKIEWICH
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2004	**FINAL** **Jan 00 – Jul 03**

4. TITLE AND SUBTITLE

MASA-CIRCA: MULTI-AGENT SELF-ADAPTIVE CONTROL FOR MISSION-CRITICAL SYSTEMS

5. FUNDING NUMBERS

C - F30602-00-C-0017
PE - 62301E
PR - J124
TA - 00
WU - 01

6. AUTHOR(S)

David J. Musliner, Robert P. Goldman, Michael J. Pelican
Kurt D. Drebsbach

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Honeywell Technology Center
3660 Technology Drive
Minneapolis MN 55418

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

AFRL/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-32

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Daniel E. Daskiewich/IFTB/(315) 330-7731 Daniel.Daskiewich@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The goal of this contract effort was to begin extending the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Multi-Agent Self Adaptive (MASA) CIRCA. CIRCA is a coarse-grained architecture designed to control autonomous systems which require both intelligent, deliberative planning activity and highly reliable, hard real-time reaction to safety threats. The MASA-CIRCA project extended this architecture with the ability to reason accurately about its own real-time behavior, and adapt that behavior in response to performance feedback. Major issues investigated during this project include formally verifying real-time control plans, dynamically decomposing long-term plans into sub goals, and building real-time control plans using probabilistic information to reason about most-likely states first. We provided digital video demonstrations of these features, with MASA-CIRCA operating in a combat oriented multi-aircraft flight simulation domain.

14. SUBJECT TERMS

Software agents, real time planning, autonomous systems

15. NUMBER OF PAGES 68

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

Preface

The work reported here was conducted by the Honeywell Technology Center, Minneapolis, MN, during the period January 20, 2000 through July 31, 2003 under Air Force Research Laboratory contract F30602-00-C-0017. Mr. Dan Daskiewich was the AFRL project officer for the contract.

This report is intended to provide an overview of the project research. This report makes numerous references to more-detailed publications that have been generated during the project. All of these publications are included on this publication disk, as well as on WWW sites listed in the bibliographic information. If you choose to read the HTML version of this report, it contains active hyperlinks that will lead directly from each citation to the corresponding published paper.

MASA-CIRCA: Multi-Agent Self-Adaptive Control for Mission-Critical Systems

Abstract

This is the final report for the Defense Advanced Research Projects Agency (DARPA) contract F30602-00-C-0017 entitled "MASA-CIRCA: Multi-Agent Self-Adaptive Control for Mission-Critical Systems." The goal of this contract effort was to begin extending the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Multi-Agent SA-CIRCA (MASA-CIRCA). CIRCA is a coarse-grain architecture designed to control autonomous systems which require both intelligent, deliberative planning activity and highly reliable, hard-real-time reactions to safety threats. CIRCA allows systems to provide performance guarantees that ensure they will remain safe and accomplish mission-critical goals while also intelligently pursuing long-term, non-critical goals. The MASA-CIRCA project extended this architecture with the ability to reason accurately about its own real-time behavior, and adapt that behavior in response to performance feedback. Major issues investigated during this project include formally verifying real-time control plans, dynamically decomposing long-term plans into subgoals, and building real-time control plans using probabilistic information to reason about most-likely states first.

The primary technical products of this research project include two versions of CIRCA's controller-synthesis (or planning) algorithm. The first version, developed by Honeywell, automatically generates reactive control plans and verifies their correctness using formal model-checking methods. The second version, developed by the University of Michigan on subcontract, does not use model checking to verify its plans, but incorporates a novel form of probabilistic reasoning and a new multi-agent negotiation protocol to restrict its planning effort to the most relevant future system states. Technical products also include a completely new Adaptive Mission Planner that uses novel deliberation scheduling strategies to manage the problem-solving performed by MASA-CIRCA, and uses Contract-Net style negotiation to coordinate tasks between multiple MASA-CIRCA agents. We provide digital video demonstrations of these features, with MASA-CIRCA operating in a combat-oriented multi-aircraft flight simulation domain.

1. Introduction

This is the final report for the Defense Advanced Research Projects Agency (DARPA) contract F30602-00-C-0017 entitled "MASA-CIRCA: Multi-Agent Self-Adaptive Control for Mission-Critical Systems." The goal of this contract effort was to extend the Cooperative Intelligent Real-Time Control Architecture (CIRCA) with abilities to automatically monitor its own performance and adapt in real-time, forming Multi-Agent Self-Adaptive Cooperative Intelligent Real-Time Control Architecture (MASA-CIRCA). CIRCA is a coarse-grain architecture designed to control autonomous systems which require both intelligent, deliberative planning activity and highly reliable, hard-real-time reactions to safety threats. CIRCA allows systems to provide performance guarantees that ensure they will remain safe and accomplish mission-critical goals while also intelligently pursuing long-term, non-critical goals. The MASA-CIRCA project took several steps towards extending this architecture with the ability to reason accurately about its own real-time behavior, adapt that behavior in response to performance feedback, and communicate between agents to coordinate overall team activities.

Major issues investigated by Honeywell during this project include:

Plan Verification — MASA-CIRCA builds plans that can provide performance guarantees of system safety; these guarantees are ensured using formal model checking methods. Section 4 discusses our research into *incrementally* verifying these plans, a patent-pending approach that provides significant performance improvements.

Deliberation Scheduling — For MASA-CIRCA deliberation scheduling is the task of deciding what problems the Adaptive Mission Planner (AMP) and Controller Synthesis Module (CSM) modules should be working on at any time. Most importantly, time-consuming planning and scheduling processes must be managed to ensure that the best possible control plans are built throughout the mission. Section 11 describes our decision-theoretic approach in more detail.

Multi-agent Negotiation — Each MASA-CIRCA agent actively coordinates with other MASA-CIRCA agents in a cooperative team. Section 10 describes this capability, and Section 13.1 describes one demonstration of inter-agent negotiation.

Performance Monitoring — MASA-CIRCA monitors the execution of its plan to ensure that progress is being made towards its goals, and that unexpected situations (e.g., unexpected threats) are recognized. Section 13.4 details a demonstration of this performance monitoring behavior and the subsequent adaptations (replanning) that MASA-CIRCA performs automatically.

Section 13 describes the simulated Unmanned Aerial Vehicle (UAV) demonstrations we developed to illustrate these behaviors of MASA-CIRCA in real-time, mission-critical environments.

The University of Michigan team had, as its emphasis, issues of negotiation among components within a CIRCA agent (specifically, negotiation between the AMP and the

CSM) and between CIRCA agents in order to approximately optimize resource scheduling on the agents' execution platforms. The assumption underlying this work was that the demands placed on an execution platform might outstrip its available resources.

For example, an Unmanned Combat Air Vehicle (UCAV) might be incapable of monitoring for all conceivable threats frequently enough to guarantee a desired level of safety. When this occurs, negotiation is necessary. The CSM should request from the AMP a less demanding control problem, providing to the AMP any guidelines it can about promising problem relaxations. The AMP should use its higher-level perspective to revise and resubmit a new controller synthesis problem to the CSM. Moreover, the AMP and CSM on one CIRCA platform can potentially negotiate with their counterparts on another platform to similarly determine ways of simplifying the controller synthesis problem. The efforts at the University of Michigan developed algorithms and protocols for these purposes.

The results of this work can be summarized as follows:

Probabilistic State Space Planner — A version of the State Space Planner that is at the heart of the Controller Synthesis Module has been extended to model time-dependent transition probabilities. In the resulting Probabilistic State Space Planner (PSSP), we have developed efficient techniques to use the probabilistic information to estimate probabilities of reaching different states of the world. When too many real-time controller reactions need to be scheduled, the state probabilities permit informed tradeoffs, where the least likely reactions to be needed can be preferentially pruned until the remaining ones can be scheduled. This leads to a synthesized controller that makes probabilistic safety guarantees. The probability thresholds needed for scheduling, and hence the risk incurred, can be reflected back to the AMP, which can use that information to revise the control synthesis problem. In this way, the AMP and the PSSP negotiate to arrive at a synthesized controller that is schedulable and meets mission needs. Section 6 describes this research in more detail.

Resource-Driven Subgoaling — The State Space Planner has been augmented with algorithms that inspect the resulting state reachability graph to identify ways of breaking the larger graph down into more tractable subgraphs. If the larger graph requires more real-time reactions than can be scheduled, breaking it into a sequence of phases, where the transition from one phase to the next is under the CIRCA agent's control, can permit the desired level of safety guarantees. The new algorithms incrementally cluster states that must belong in the same phase together, and identify action transitions that lead between phases. These candidate phases are then passed back to the AMP, which can decide whether to adopt the potential mission phasing, and if so can request smaller (phase) synthesis controller problems to be solved in a negotiated manner. Section 7 describes this research in more detail.

Constrained Markov Decision Processes — In recent work, the controller synthesis problem has be formulated in terms of Constrained Markov Decision Problems (CMDPs), where the constraints are on the resources available for executing policies.

A variety of different resource constraints have been studied. The most detailed investigation has been into extending prior work on CMDPs to explicitly model the probability of exceeding a resource constraint, and to generate control policies that are probabilistically guaranteed not to exceed the resource constraint. Section 8 describes this research in more detail.

Communication to Reduce Uncertainty — During controller synthesis, the greater the number of transitions that could need preempting, the harder it is to schedule all of the necessary reactions (Test-Action Pairs (TAPs)) frequently enough. In a setting involving multiple CIRCA agents, one way of simplifying the controller synthesis problem faced by each is through the exchange of information and the negotiation over action choices made by each. Specifically, one CIRCA agent can inform another about which reaction it will execute in a particular situation, meaning that the other agent need not be prepared for the repercussions of every possible reaction that the other might have taken. By selectively communicating about aspects of the controllers they have each synthesized, they can help each other remove unnecessary portions of their controllers, and therefore increase the effectiveness and safety assured by the controllers ultimately generated. Section 9 describes this research in more detail.

2. Overview of CIRCA [1]

CIRCA is designed to support both hard real-time response guarantees and unrestricted Artificial Intelligence (AI) methods that can guide those real-time responses. Figure 1 illustrates the architecture, in which the AMP and CSM reason about high-level problems that require their powerful but potentially unbounded planning methods, while a separate Real-Time Subsystem (RTS) reactively executes the automatically-generated plans and enforces guaranteed response times. The AMP and CSM modules cooperate to develop executable reaction plans that will assure system safety and attempt to achieve system goals when interpreted by the RTS.

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics and simulated unmanned aircraft (UAVs). A UAV example will be discussed in detail in Section 3. To introduce the key CIRCA concepts in this section, we draw examples from the domain illustrated by Figure 2, in which CIRCA controls a simulated Puma robot arm that must pack parts arriving on a conveyor belt into a nearby box. The parts can have several shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not initially know how to pack all of the possible types of parts— it may have to perform some computation to derive an appropriate box-packing strategy. The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system must push the button next to the light before a fixed deadline.

[1] This section and the next are drawn largely from prior material, including the prior SA-CIRCA project final report.

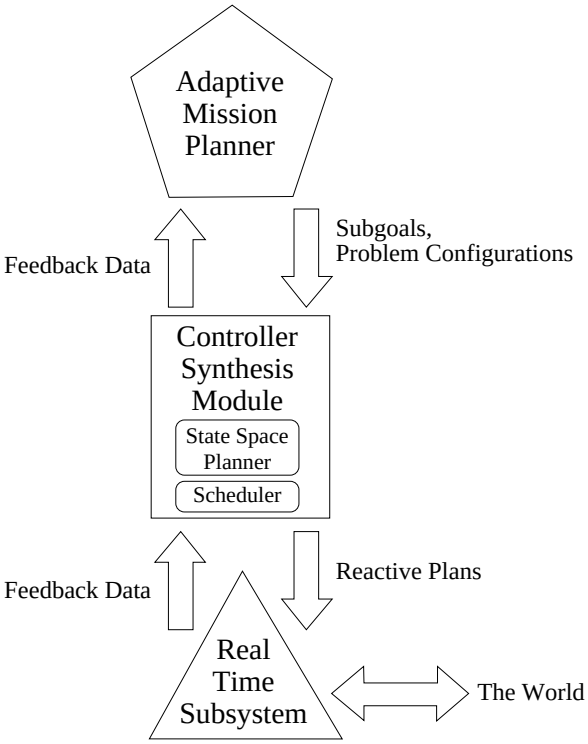


Figure 1. The CIRCA architecture combines intelligent planning and adaptation with real-time performance guarantees.

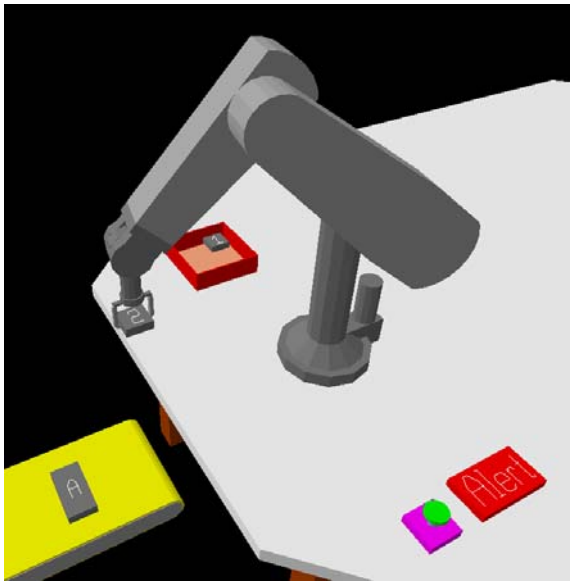


Figure 2. The simulated Puma robot arm domain.

```
EVENT emergency-alert                          ;; Emergency light goes on
      PRECONDS: ((emergency nil))
      POSTCONDS: ((emergency T))

TEMPORAL emergency-failure                     ;; Fail if don't attend to
      PRECONDS: ((emergency T))                ;; light by deadline
      POSTCONDS: ((failure T))
      MIN-DELAY: 30 [seconds]

ACTION push-emergency-button
      PRECONDS: ((part-in-gripper nil))
      POSTCONDS: ((emergency nil) (robot-position over-button))
      WORST-CASE-EXEC-TIME: 2.0 [seconds]
```

Figure 3. Example transition descriptions given to CIRCA's planner.

In this domain, CIRCA's planning and execution subsystems operate in parallel. The CSM reasons about an internal model of the world and dynamically programs the RTS with a planned set of reactions. While the RTS is executing those reactions, ensuring that the system avoids failure, the AMP and CSM are able to continue executing heuristic planning methods to find the next appropriate set of reactions. For example, the AMP may derive a new box-packing algorithm that can handle a new type of arriving part. The derivation of this new algorithm does not need to meet a hard deadline, because the reactions concurrently executing on the RTS will continue handling all arriving parts, just stacking unfamiliar ones on a nearby table temporarily. When the new box-packing algorithm has been developed and integrated with additional reactions that prevent failure, the new schedule of reactions can be downloaded to the RTS.

CIRCA's State Space Planner builds reaction plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans [27]. To describe a domain to CIRCA, the user inputs a set of transition descriptions that implicitly define the set of reachable states. For example, Figure 3 illustrates several transitions used in the Puma domain. These transitions are of three types:

Action transitions represent actions performed by the RTS.
Temporal transitions represent the progression of time and continuous processes.
Event transitions represent world occurrences as instantaneous state changes.

The SSP plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. The SSP assigns to each reachable state either an action transition or `no-op`. Actions are selected to *preempt* transitions that lead to failure states and to drive the system towards states that satisfy as many goal propositions as possible. A planned action preempts a temporal transition when the action will definitely occur before

```
#<TAP 10>
        Tests   : (AND (PART_IN_GRIPPER NIL) (EMERGENCY T))
        Acts    : push_emergency_button
        Max-per : 9984774
        Runtime : 2520010
#<TAP 9>
        Tests   : (AND
                    (PART_IN_GRIPPER NIL)
                    (EMERGENCY NIL)
                    (PART_ON_CONVEYOR T)
                    (NOT (TYPE_OF_CONVEYOR_PART SQUARE)))
        Acts    : pickup_unknown_part_from_conveyor
        Max-per : 12029856
        Runtime : 3540010
#<TAP 8>
        Tests   : (AND
                    (TYPE_OF_CONVEYOR_PART SQUARE)
                    (PART_IN_GRIPPER NIL)
                    (EMERGENCY NIL))
        Acts    : pickup_known_part_from_conveyor
        Max-per : 12029856
        Runtime : 3520010
```

Figure 4. Sample output from the TAP compiler.

the temporal transition could possibly occur. The assignment of actions determines the topology of the NFA (and so the set of reachable states): preemption of temporal transitions removes edges and assignment of actions adds them. System safety is guaranteed by planning action transitions that preempt *all* transitions to failure, making the failure state unreachable [27]. It is this ability to build plans that guarantee the correctness and timeliness of safety-preserving reactions that makes CIRCA suited to mission-critical applications in hard real-time domains.

The NFA is translated into a memoryless controller for downloading to the RTS. This is done through a two-step process. First, the action assignments in the NFA are compiled into a set of *Test-Action Pairs* (TAPs). The tests are a set of boolean expressions that distinguish between states where a particular action is and is not to be executed. Each TAP's test expression is derived by examining all of the planned actions and finding a logical expression that distinguishes between the states in which the current TAP's action is planned and the states in which other actions are planned. Some sample TAPs for the Puma domain are given in Figure 4.

Eventually, the TAPs will be downloaded to the RTS to be executed. The RTS will loop over the set of TAPs, checking each test expression and executing the corresponding action if the test is satisfied. The tests consist only of sensing the agent's environment, rather

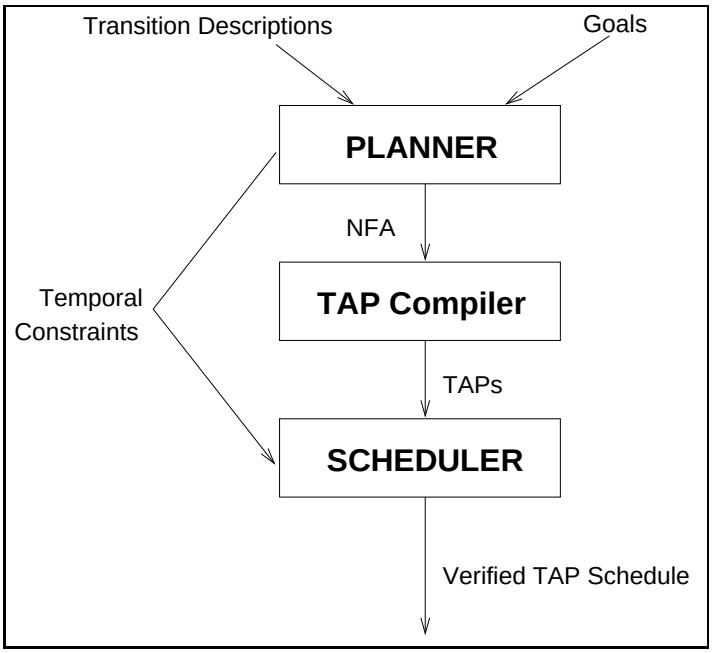


Figure 5. Summary of the CIRCA state space planning process.

than checking any internal memory, so the RTS is asynchronous and memoryless.

However, before the TAPs can be downloaded, they must be assembled into a loop that will meet all of the planned deadlines, captured as constraints on the maximum period of the TAPs (see Figure 4). This second phase of the translation process is done by the scheduler in the CSM. In this phase, CIRCA's scheduler verifies that all actions in the TAP loop will be executed quickly enough to preempt the transitions that the planner has determined need preempting. The tests and actions that the RTS can execute as part of its TAPs have associated worst-case execution times that are used to verify the schedule. If the scheduling does not succeed, the SSP will backtrack to revise the NFA, leading to a new set of TAPs and another scheduling attempt. The planning process is summarized in Figure 5.

3. Verifying State Space Plans

3.1. The CIRCA SSP

The CIRCA SSP automatically synthesizes timed discrete-event controllers for hard real-time applications. The input to the SSP is a description of a control problem in the form of environment dynamics (including uncontrollable processes and threats to system safety), actions available to the controller, and goals to be realized. The SSP returns a controller that is guaranteed to maintain the safety of the controlled system. The controller specifies what action should be taken for each reachable system state. The controller provides safety guarantees by meeting the timing requirements of the control problem; these timing requirements are inferred from the model of the uncontrollable processes that threaten the system. To determine that these timing requirements are met, our algorithm consults a model-checker for real-time automata. This model-checking is done on an

incremental basis, as the controller is built.

For example, Figure 6 contains the transition descriptions for a simple UAV control problem. The transitions describe a problem in which a UAV is attempting to follow a normal flight path (hence the `*goals*` statement). However, at any time during its flight, the UAV might be tracked by enemy radar. Some time after the initial tracking, a Surface-to-Air Missile (SAM) may be launched. If no countermeasures are taken, that SAM may destroy the UAV after at least a certain minimum amount of time has passed (e.g., the minimum flight time of the missile). The UAV has available to it some evasive maneuvers that will cause the SAM to miss the UAV, if the UAV initiates its maneuvers quickly enough. Also, since the maneuvers divert the UAV from its nominal trajectory, the UAV should end its evasive behavior whenever possible.

Figure 7 shows the state space resulting from a simple timed controller design that will preserve the safety of the UAV. In the initial state, labeled "State 17" and shown as a shaded oval, the UAV is on its normal trajectory and has no indication of a radar-guided missile tracking it. This is a desirable state, so the controller will make no effort to leave it. However, at any time, a radar threat could occur, moving the system into state 16. The controller will react to this threat by taking evasive action, and maintaining the evasive maneuvers until the missile has been avoided (i.e., until the system has entered state 24). At this time the threat has been neutralized, and the system is free to return to its normal flight path. This controller was automatically generated by CIRCA, and the state diagram was generated from CIRCA data structures by the daVinci program [10].

There are several important aspects to note about this example state space model, or finite automaton. First, note that the automaton contains loops: the UAV may be threatened by more than one missile, and will remain in (or re-start) evasive maneuvers as long as it is threatened. Second, observe that time is not an explicit part of the state representation. This is critical to the compact representation of looping plans; if we included time in the state representation, then loops would not occur and persistent reactive control against an unpredictable or adversarial world would explode the state space. Instead, our automaton neatly encodes the continously-reactive behavior of the UAV in a compact, efficient, and automatically-generated form. Of course, the transitions do have temporal semantics, as described in Figure 6.

The SSP's temporal model was carefully designed to support reasoning about system safety with only a minimal amount of temporal information, thus limiting the complexity of the automata model. We associate with each transition a set of bounds on the time (Δ) which the system must dwell in the transition's source state before the transition could possibly occur. The model includes four different types of transitions:

Temporal Transitions — Drawn as double arrows, temporal transitions represent uncertain processes that may lead to change, but only after at least some minimum amount of time has passed ($\Delta \geq min\Delta$). The only temporal transitions in our simple UAV example lead to failure, and are not shown in Figure 7 because the

```
(setf *goals* '((path normal)))

;; Radar-guided missile threats can occur at any time.
(make-instance 'event
        :name "radar_threat"
        :preconds '((radar_missile_tracking F))
        :postconds '((radar_missile_tracking T)))

;; You die if don't defeat a threat by 1200 time units.
(make-instance 'temporal
        :name "radar_threat_kills_you"
        :preconds '((radar_missile_tracking T))
        :postconds '((failure T))
        :min-delay 1200)

;; It takes no more than 10 time units to start evasives.
(make-instance 'action
        :name "begin_evasive"
        :preconds '((path normal))
        :postconds '((path evasive))
        :max-delay 10)

;; We defeat missile in between 250 and 400 time units.
(make-instance 'reliable-temporal
        :name "evade_radar_missile"
        :preconds '((radar_missile_tracking T)
                    (path evasive))
        :postconds '((radar_missile_tracking F))
        :delay (make-range 250 400))

;; It takes no more than 10 time units to end evasives.
(make-instance 'action
        :name "end_evasive"
        :preconds '((path evasive))
        :postconds '((path normal))
        :max-delay 10)
```

Figure 6. A simple domain description for a UAV threatened by radar-guided missiles.

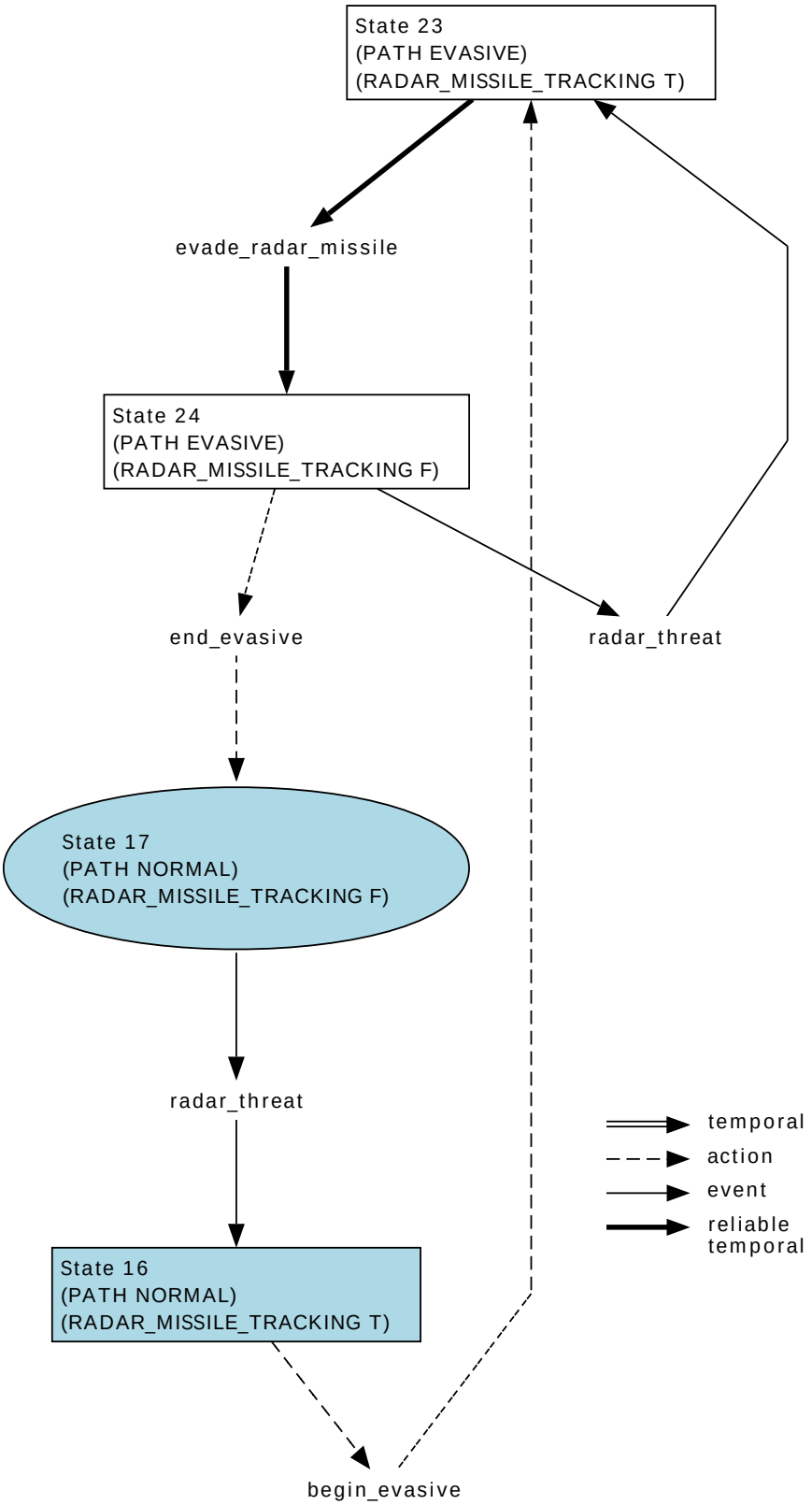


Figure 7. Simple UAV controller for evading radar-guided SAM threats.

safety-preserving controller design makes failure unreachable.

Event Transitions — Drawn as single arrows, event transitions represent instantaneous transitions that are out of our control, and may happen any time their preconditions are satisfied. They are essentially the same as temporal transitions with a $min\Delta$ of zero.

Action Transitions — Drawn as dashed arrows, action transitions represent processes that are guaranteed to occur before the system has dwelled a certain amount of time in the source state. That is, action transitions will definitely occur before Δ reaches an upper bound $max\Delta$.

Reliable Temporal Transitions — Drawn as bold single arrows, reliable temporal transitions represent processes that are guaranteed to occur, if given enough time. They have both lower and upper bounds on the dwell time the system must stay in the source state before the reliable temporal transition will occur ($min\Delta < \Delta < max\Delta$).

Using this information, the SSP reasons about one key temporal relationship: *preemption*. A transition t is preempted if and only if some other transition u from the same state must definitely occur before t could possibly occur. In other words, t is preempted if and only if $max\Delta(u) < min\Delta(t)$. In our UAV example, the `radar_threat_kills_you` transition is preempted in state 16 by the action transition `begin_evasive`.

Preemption is the key temporal relationship in CIRCA models because it allows the SSP to build discrete event controllers that make certain parts of the potential system state space unreachable. By making all potential failure states unreachable, the SSP can build plans (controllers) that are guaranteed to keep the system safe, while also pursuing other less-critical goals. The goal of plan verification, discussed in the next section, is to prove that the preemptions CIRCA has planned will in fact hold true for all possible future world "trajectories" (i.e., paths through the reachable states).

Note that the `begin_evasive` action does not actually disable the `radar_threat_kills_you` transition: it simply begins the process of defeating the threat, which is represented by the reliable temporal transition `evade_radar_missile`. If we were to draw the Temporal Transitions to Failure (TTFs) in the graph of Figure 7, we'd see that the `radar_threat_kills_you` TTF is actually preempted out of both state 16 and the subsequent state 23. This is called a *dependent temporal chain*, because the amount of time left to preempt the TTF in state 23 is not the original minimum dwell time (as it was in state 16), but the original $min\Delta$ minus however much time the system may have dwelled in state 16 before transitioning to state 23. Since CIRCA reasons about worst-case circumstances, that dwell time is equivalent to the upper bound dwell time ($max\Delta$) imposed by the planned action `begin_evasive`. Hence the new $min\Delta$ in state 23 is actually $1200 - 10 = 1190$.

Thus our temporal model is actually non-Markovian: the temporal semantics of the TTF out of state 23 depend on the path the system takes to get there. Naturally, this

complicates the process of reasoning about the temporal model, and motivates our use of model checking to verify the required TTF-preemption properties.

3.2. Model Checking for Plan Verification

In order to verify that the CIRCA SSP's plans are safe, we must project what will happen when they are executed. We must determine whether the actions we have planned do, in fact, preempt all possible transitions to failure. To do so, we use techniques developed in the computer-aided verification research community; specifically we use techniques for verifying properties of *timed automata* [1].

A naive algorithm for CIRCA plan verification is easy to propose: start at the initial state(s), find all the possible successor states, and repeat. If you ever enter a failure state, the verification has failed.

The problem with this algorithm is hidden in the definition of system state. To determine the possible successor states, we must know how long transitions have been enabled. For example, to determine at state 23 whether radar_threat_kills_you happens before or after evade_radar_missile, we must know whether the former transition has been active for 1200 time units before the latter has been active for 400 (see Figure 6).

Imagine that each transition has associated with it a timer, or "clock." When the transition is enabled, that clock is reset to zero and started. When the transition is disabled, that clock is turned off. Whenever that clock goes over the lower bound on the corresponding transition, the transition may occur; the transition is guaranteed to occur before the upper bound on the transition (unless some other transition intervenes).

Thus we can characterize the full state of the controlled system by the full set of feature values and a vector of artificial clock values. For example:

flight_path = evasive
radar_missile_tracking = true
clock(evade_radar_missile) = 40
clock(radar_threat_kills_you) = 700

By comparing this state against the problem definition given in Figure 6, you may readily see that this state is safe. radar_threat_kills_you cannot take place for 500 more time units, by which time evade_radar_missile will have preempted it.

Unfortunately, the verification problem, as naively framed, is not practically solvable. Since the clocks are integer-valued,[2] the set of system states is infinitely large. However, the set of interesting values is less than infinite, since there are only a finite number of decisions that need be made. For example, all values of clock(radar_threat_kills_you) that are over 1200 are equivalent. However, the number of relevant states may still be very large.

3.2.1. Timed Automata Representation

Fortunately, researchers in computer-aided verification have found ways to compactly represent states like this for a class of finite state machines called *timed automata* [1].

[2] Although time is continuous, it may be discretized without loss of accuracy for any verification problem.

Timed automata differ in a few minor ways from SSP state machines, but SSP state machines can be translated into timed automata. Timed automata states are composed of a *location* (corresponding to an SSP state, or feature vector) and a *clock-interpretation, or vector of clock values*. All of the clocks increment synchronously, but can be independently reset to zero by selected transitions. Transitions themselves are instantaneous. Temporal constraints in timed automata take two forms: transition *guard expressions* that must be true to enable a transition, and *state invariant* expressions that must be true all the time the system remains in a particular state.

Mapping an SSP state space model into a timed automaton is a fairly simple matter of assigning different clocks to different CIRCA transitions and translating the CIRCA transition timing constraints into timed automaton clock constraints. Once this translation is complete, the timed automaton model can be passed to our model-checking code, the Real-Time Analysis (RTA) module, to determine whether failure is reachable and, if so, what path of transitions leads to failure (to guide CIRCA's intelligent backjumping).

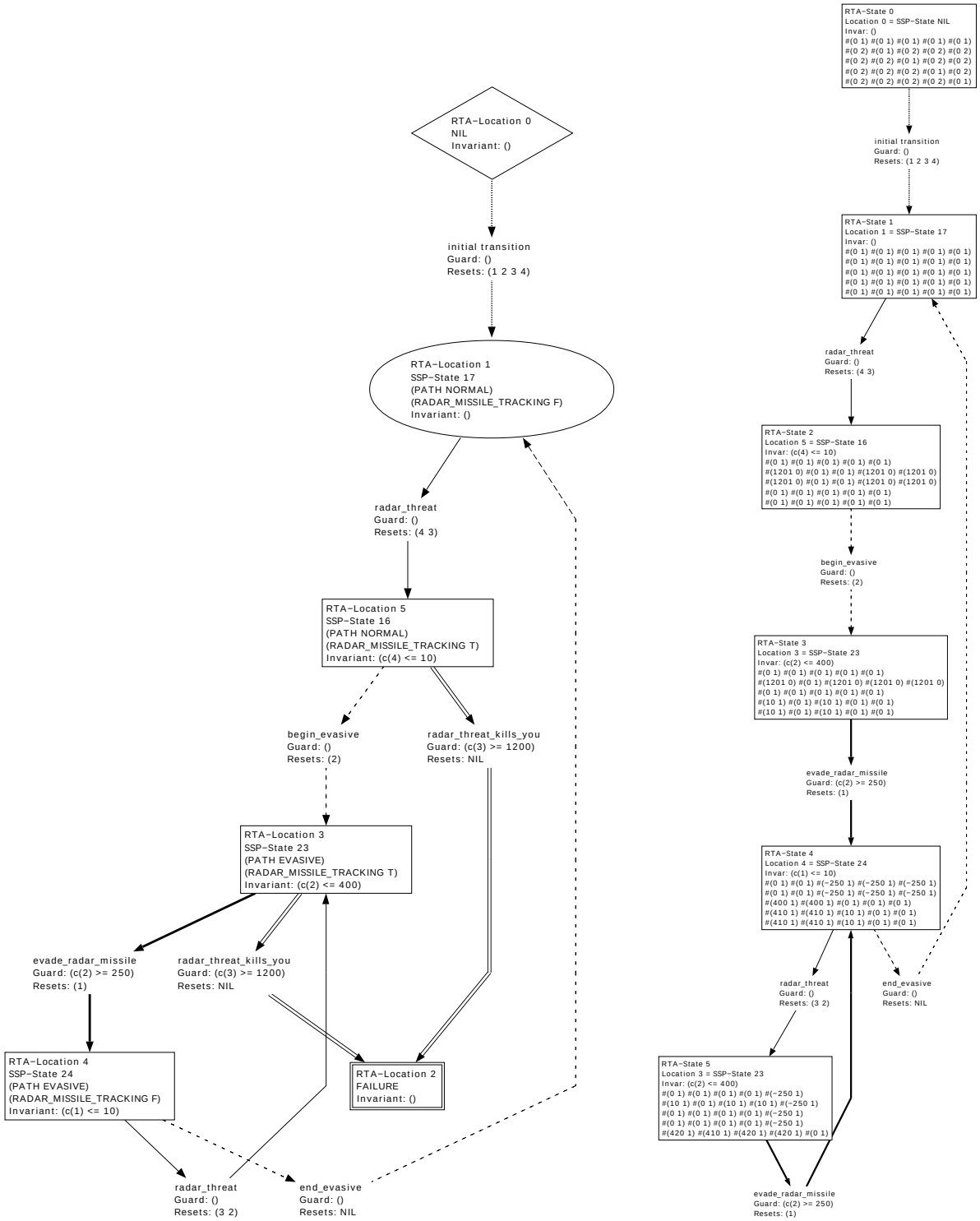
Figure 8a illustrates the RTA timed automaton that corresponds to CIRCA's solution to the UAV example of Figure 7. Briefly, the automatic translation process involves mapping each type of SSP state space transition, as follows:

Temporal Transitions — Temporal transitions require the system to dwell in a state for a certain amount of time before the transition may occur. This corresponds exactly to a transition guard expression in RTA. Thus temporal edges are each assigned a clock, and have guard expressions constraining the value of that clock to be greater than the temporal transition's minimum delay. The clock is reset by all edges entering the source state of the temporal edge, if that edge does not come from a state in which the same temporal is enabled.

Event Transitions — Event transitions can occur at any time, so they have no associated clocks and are simply unrestricted edges in the RTA graph.

Action Transitions — Action transitions place an upper bound on the time the system may dwell in the transition's source state before it necessarily will move to the transition's sink state. In our RTA model, this corresponds to an upper bound state invariant expression. Each instance of an action transition (action edge) is assigned a new clock. The clock is reset by all edges entering the source state of the action edge, if that edge does not come from a state in which the same action is enabled. The action edge itself has no guarding clock constraints; instead, the action edge's upper bound is expressed as an invariant in the edge's source state.

Reliable Temporal Transitions — Reliable temporals combine the lower-bound and upper-bound timing constraints of temporals and actions, so their RTA mapping uses transition guards to represent the lower bounds and state invariants to represent the upper bounds.



(a) Input finite automata. (b) Clock-zone expansion.

Figure 8. The input model and clock zone analysis for the example UAV domain.

3.2.2. Efficient Model Checking

The critical concept for taming the complexity of timed automaton verification is an equivalence relation ("region equivalence") between system states [1]. This equivalence relation makes use of the intuition that all values for a given clock are equivalent above a maximum value (the largest constant the clock is ever compared to). Furthermore, since we are only concerned with the reachability of various states, the actual values of different clocks in a state are not as important as their *relative* values. Because the clocks are all notionally incremented at the same rate, the relationships between the clock values upon entry to a state is sufficient to determine which outgoing transitions are possible: a clock that is behind another cannot catch up (within a state). Based on this equivalence relation, it can be shown that any timed automaton (SSP plan) has only a finite number of states.[3] Therefore, the problem of determining reachability (SSP plan verification) is decidable.

A further optimization is possible, to make verification practical. The key intuition behind this optimization is that all reachability questions hinge on pairwise comparisons between clock values. In order to determine whether or not one transition can occur, we compare a single clock against a constant. To determine whether one transition occurs before another, we only need to determine which will reach its associated constant first. To answer this question, we only need to know the *difference* between pairs of clock values (since the clock values increase at the same rate).

Therefore, we can compactly represent clock regions using a *difference-bound matrix* [4] whose entries represent bounds on the difference between pairs of clocks and between single clocks and a dummy clock whose value is always zero. Difference-bound matrices have two advantages. First, they provide a compact representation for equivalence classes of clock-states in timed automata. Second, they also have a canonical form, derived using any standard all-pairs shortest-path algorithm [4]. Putting the associated difference-bound matrices into canonical form makes it easy to determine when two automaton states are equivalent. Recognizing equivalent states is, in turn, necessary in order for reachability search to terminate.

Figure 8b illustrates the reachability verification of the SSP plan given in Figure 6, optimized by the use of difference-bound matrices. Space limits preclude us from describing the difference bound notation in detail. However, a simple examination of Figure 8b shows one notable aspect of the RTA verification: there are two RTA states (3 and 5) that correspond to the SSP state 23. That is, the RTA algorithm has recognized the distinction between the two routes into SSP state 23 (see Figure 7) as being a temporally significant difference. The temporal transition to failure from state 23 will have different amounts of time left on its clock depending on whether we enter from state 16, where it was already enabled, or state 24, where it was not enabled (see Figure 8a, RTA-locations 5 and 4). Thus the RTA algorithm is unrolling the important paths through dependent temporal

[3] More precisely, there are only a finite number of state equivalence classes, and state equivalence classes are sufficient to determine reachability.

chains, checking reachability of failure by removing the original non-Markovian temporal semantics.

For even modest sized SSP problems, the computational costs of verification can be prohibitive. In the next section, we discuss how to dramatically reduce those costs by performing the verification incrementally, as the planning process proceeds.

4. Incremental Verification

4.1. Background

CIRCA uses model-checking verifiers during its planning process to ensure that its plans will in fact satisfy their performance guarantees. CIRCA verifies partial plans as it is generating them, so that it does not try to complete partial plans that are already unsafe. Using verification inside the planning loop also makes CIRCA's controller synthesis more efficient. First, by relying on the verifier to consider every possible sequence of states and transition delays, the planner can reduce its search space considerably by considering time-abstract world states. Second, the counter-examples (in the form of state and transition sequences) are used to inform the backtracking done by the planning search. Third, the planner's time abstraction alone is insufficient to accurately determine which world states in the search space are reachable, so without a verifier it must use a liberal notion of reachability to maintain safety (sometimes considering some states that are not, in fact, reachable). The verifier can determine reachability exactly and returns this information to the search, which can avoid wasting computational effort planning for unreachable states.

These benefits have a cost: in general, verification time dominates planning time by a wide margin in CIRCA planning problems. There are several possible paths to improvement:

- Code optimization: Over the years, we have invested substantial effort in low-level optimization of the algorithms and data structures relevant to the inner-loop of plan verification.
- Less frequent verification: We have considered (but not yet implemented) reducing the frequency of verification during planning. In this case, the planner will sometimes do unfruitful work, but this wasted effort may be very cheap relative to the cost of verification at every iteration.
- Approximate verification: The model-checking community has developed useful algorithms for approximate model-checking, which are more efficient than the exact equivalents. It would be interesting to pursue this direction for CIRCA, but understanding the resulting loss of completeness would be a challenge.
- Temporal inclusion: It is possible to identify verifier states which are included by the temporal bounds of an existing verifier state. At the moment, we only match verifier states if they have identical difference-bound matrices.

- Re-using partial computations: The partial plans developed by CIRCA change very little between verification iterations. We could save a lot of computational effort by saving the results of previous verifications and only verifying the parts of the model that have changed.

We have pursued this final approach, which we call "Incremental Verification" with great success.

4.2. Intuition

During the planning process the planner and verifier maintain separate graphs representing possible future world states. For the purposes of this discussion, the principal difference between the graph used by the planner and that used by the verifier is the representation of time. The planner ignores time, except as it is reflected in possible sequences of states and transitions. The verifier dynamically derives a compact representation of the relevant temporal state by computing bounds on the length of time each of the enabled transitions out of a state have been active. Accordingly, each planner state corresponds to a set of states in the verifier's graph, i.e., the set of states with the same feature values, but different temporal characteristics. For example, although the planner knows only that there is a heat-seeking missile heading for the aircraft, the verifier must know whether there is enough time remaining for a countermeasure to have an effect. If T is the time required for the countermeasure, it will have one state in which T or more seconds remain before the missile is expected to strike the aircraft and another in which less than T seconds remain.

Furthermore, the verifier treats any frontier state in the planner's graph (i.e., any unplanned state) as a safe, sink state.

In order to continue the verification process incrementally, then, the verifier must (1) identify states affected by the last planning step and (2) update those states to reflect the modifications made by the planner. To support (2), the verifier must also maintain some additional information for the frontier states, since they are now continuable.

4.3. Implementation

As described in the previous section, the incremental verification algorithms differ substantially from the previous versions in three ways:

- the treatment of frontier states,
- the identification of states affected by a planning step, and
- the updating of states affected by a change to the plan.

4.3.1. Treatment of Frontier States

In prior implementations, frontier states in the verifier graph (i.e., states that correspond to unplanned states in the planner's graph) could be treated as safe sink states by the

verifier. If the frontier states are believed to be failure states, then the planner would have declared failure before calling the verifier. In the incremental version, these states must be treated as potentially continuable and sufficient information from creation time should be stored for later use.

For the incremental case, the relevant pseudo-code is:

1. Create new verifier state, VS.
2. Lookup corresponding planner state, PS.
3. If PS is planned, continue as before.
4. If PS is not planned:
   (a) Create a state continuation, C, containing an index to PS, the preceding verifier state, and the incoming transition.
   (b) Put C on the open list.

For the sake of efficiency, the continuation also contains a partially computed difference bound matrix. Not surprisingly, the temporal extent of a verifier state depends on the preceding state, all of the preceding state's outgoing transitions, the transition leading into the state, and all of the transitions leaving the state. Since the transitions leaving the state will not be completely known until the corresponding planner state has been assigned an action, the difference-bound matrix cannot be computed. However, the computation of the difference-bound matrix can be structured so that the factors that depend on the preceding state can be used to create an intermediate result in common for all of its successors. This partial computation is performed at the creation of the continuation and stored in the continuation.

4.3.2. Identifying Affected States

Identifying the verifier states affected by a change in the plan is straightforward, if planner state indices are stored with continuations. When an action is assigned to a planner state, any continuations corresponding to that planner state will need to be updated. Any successors to the continuations will be either:

- a new continuation (i.e., the planner state is unplanned), or
- a new verifier state (i.e., the exact combination of difference-bound matrix and feature values has not been encountered before, or
- an existing verifier state.

In the first case, the verifier will create a new continuation. In the second, it will continue the expansion of that branch of the graph. In the third, it will close the state and continue search elsewhere. In any of these cases, it is not necessary to pre-compute whether the

states are affected by the plan change. The normal behavior of the verifier algorithm is sufficient.

In the case of splits, only the split state must be identified before the update is performed. Other affected states will be modified in the course of the state update.

4.3.3. Updating Affected States

Updating the verifier states after an action assignment is a simple matter of continuing the computations that were suspended at the creation of the continuation. Once an action has been assigned by the planner, it is possible to find the set of transitions leaving the corresponding verifier states. There is one slight complication. The semantics of the verifier graph require that the transitions which reset (i.e., whose elapsed time clocks are set to zero) depends on the action chosen in that state. This list of reset clocks is computed and the normal state creation machinery is engaged.

In the case of states split by the planner (i.e., made less abstract), updating the affected states is slightly different. In this case, multiple new continuations must be created for each of the affected states, and the planner-state indices properly updated.

4.4. Results

We have merged the incremental verification code into our baseline and are currently deriving performance results. The results are encouraging: for at least one domain with relatively complex verification problems but relatively little backtracking, the incremental verification system runs several hundred times faster than the non-incremental verifier. Incremental verification has passed all of our regression tests, dramatically improving performance on many of the test domains.

The older RTA verifier algorithm was made into an incremental version that retains verification information during forward search, allowing forward growth of the CSM's state space to be verified by simply extending the traces from prior verification runs. When the search algorithm backtracks, the cached verification (trace) information is discarded and the verifier starts from scratch again.

This incremental behavior has provided dramatic speedups in some cases, especially when a large number of states are involved with relatively little backtracking required. In one case, planning time was reduced from over five minutes to under 20 seconds. However, the results are not uniformly positive: it appears that the culprit paths returned by the incremental version can correctly be different than those returned by the non-incremental version, and in some cases this leads the search into different paths that are less effective. We are investigating in more depth to understand this behavior.

The incremental version of the more powerful CIRCA-Specific Verifier (CSV) provides similarly impressive results on many domains. Again, the results are not uniformly positive because of different culprits.

4.5. Performance versus Kronos

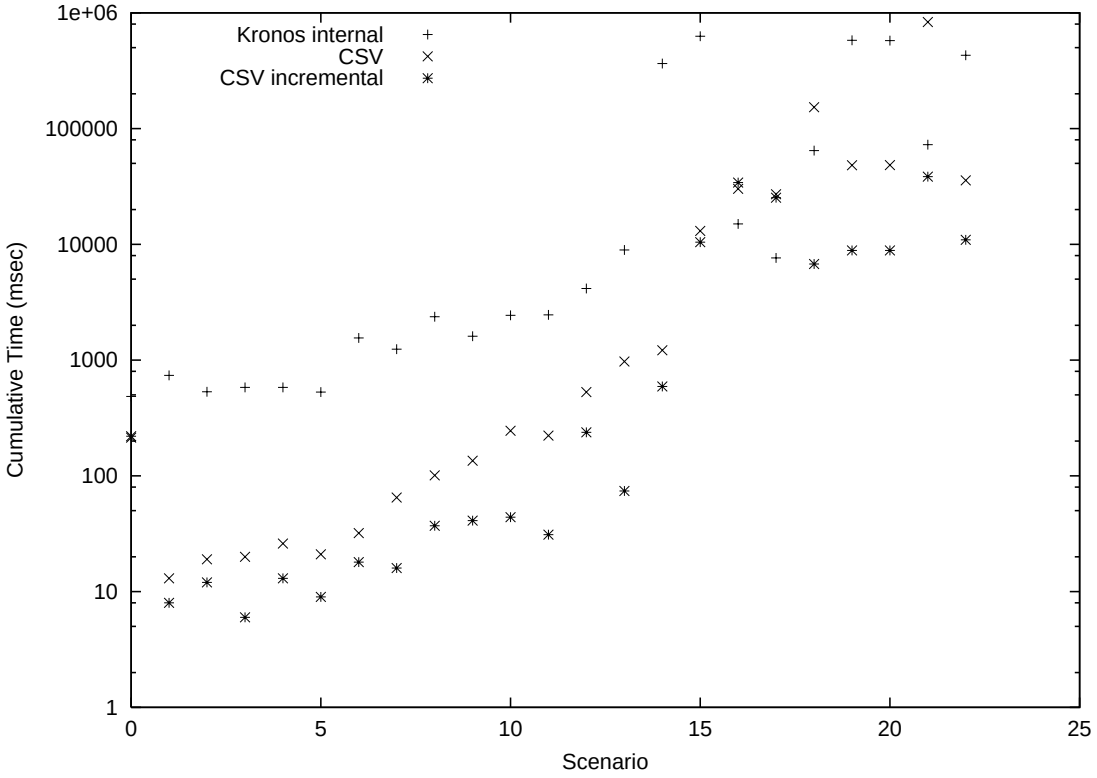


Figure 9. The incremental CIRCA-Specific Verifier is faster than Kronos on all but two domains.

Since Kronos can perform more general verification tasks than CSV, this comparison cannot be completely aligned. However, for the types of verification we need in CIRCA, Kronos and CSV can operate on very similar models. After communications with the Verimag researchers, we were able to fix problems with an earlier version of our "multi-model" Kronos interface. This is distinct from the regular Kronos interface, which we have used without problems for several years. The multi-model interface is a more accurate model of the verification problem addressed by CSV, and thus allows us to directly compare the verification results of Kronos and CSV. As illustrated in Figure 9, the incremental CSV was able to significantly outperform Kronos, requiring one to two orders of magnitude less verification time across 21 of 23 different test domains (our regression testing suite). We are currently investigating the characteristics of the remaining two domains to see why CSV did badly. In addition, we hope to obtain the highly-optimized Kronos Difference Bound Matrix (DBM) libraries and reap further speedups.

4.6. Patent

In July 2001, Honeywell management decided to pursue a patent submission for the invention disclosure entitled "Incremental Automata Verification Technique". A copy of the invention disclosure was filed with AFRL. In August 2001, our patent application was drafted. We then reviewed the application and submitting comments to the lawyers, who

will develop and circulate a second draft prior to submitting the application. In September 2001, we received the second draft of our patent application, and formulated the final round of comments before submission in October. In October 2001, our patent application was submitted after a third round of revisions. As of July 2003, our patent application is still in process.

4.6.1. Future Work

While incremental verification has already resulted in great performance improvements, further improvements should be possible. For example, the current implementation of incremental verification only uses previous computations when the planner has extended the current plan by adding a transition or splitting an abstract state. When the planner is forced to backtrack, the verification starts from scratch. It is certainly possible, and probably profitable, to modify the verification graph to reflect the planner's backtracking, and restart verification from there.

5. Partial Dynamic Abstraction and Heuristic Improvements

MASA-CIRCA uses a state space search formulation called *Dynamic Abstraction Planning* (DAP) to construct control policies [12]. DAP was developed in an effort to significantly reduce the number of explicitly represented states within the search space, thus reducing the overall search time and memory overhead. DAP works by starting with a universally abstract state, one that subsumes all possible non-failure states within the search space, and analyzing it to determine if there are any possible transitions to a failure state. If such a transition exists, DAP chooses a predicate to *split* on. Splitting on a predicate produces a set of states, one for each value in the split predicate's domain. For example, assume a predicate `A-in-room`, which tracks the location of object A, can accept the values `room1`, `room2`, and `room3`. An abstract state may not include the `A-in-room` predicate, and would thus be considered to match (subsume) all of the possible locations of object A. If the SSP splits that state on the `A-in-room` predicate, it replaces the original state with three new states with the following ground values: `(A-in-room room1)`, `(A-in-room room2)`, and `(A-in-room room3)`. The SSP heuristic guides the choice of when to split on different predicates.

DAP's development was motivated by a desire to reduce the number of states generated in the process of creating a controller, and it does this by only splitting on predicates that are relevant to avoiding failure or achieving goals. However, we noted that DAP does not perform well on a benchmark robot navigation and object-delivery domain [15, 33]. The domain consists of a set of rooms connected by doors, a set of objects, and a robot that is capable of opening doors, moving from room to room through open doors, and picking up and putting down the objects. The goal is to move the objects from their initial positions into specified goal positions and to leave the robot in a predefined position. The domain differs from its classical form with the addition of a random process that closes the doors.

An analysis of DAP's performance on this domain found that the domains of the predicate were relatively large, producing many new states with each split done by DAP. As a consequence, the search space grows rapidly and quickly overwhelms the search engine. This motivated research in developing a search method that was insensitive to the size of the predicate domains.

5.1. Partial Dynamic Abstraction Planning (pDAP)

DAP's downfall is the aggressiveness by which it splits on a predicate, generating not only the states with predicate value of interest, but a state for every other potentially-irrelevant predicate value as well. This observation led to the development of a least-commitment form of DAP called pDAP, the Partial Dynamic Abstraction Planner. pDAP avoids the proliferation of states that affected DAP when working on the robot domain through a modified splitting method. Instead of replacing the original state with a set of states for each legal ground value of the predicate, pDAP replaces the original state with two states: one with a ground version of the predicate with the relevant value; the other an abstract state whose split predicate matches every other value of the predicate.

pDAP produced significant improvements in solving problems in the robot domain. However, it too had unexpected difficult finding solutions within this domain. Analysis showed that pDAP, like DAP, was also producing a large number of irrelevant states. This time, the production of irrelevant states could not be attributed to an overly aggressive splitting method, but could only be due to a weak heuristic guiding the splitting process. As a consequence, a revised domain-independent heuristic was developed to guide the search.

5.2. Regression Graph Heuristic

The heuristic developed is based on McDermott's Regression Graph heuristic [22]. The process works by taking the goal of the planning problem, breaking the goal into its component literals and regressing the literals through actions that achieve them. The process then recurses on the preconditions of the action. Identical literals are joined to produce a possibly cyclic graph that is fully specified. Once the regression graph has been expanded, literals true in the current state are marked and path lengths from the goal literals to current state literals are calculated. An operator on the shortest path is then selected by the heuristic.

MASA-CIRCA uses this heuristic in essentially its original form when it is looking for applicable actions. However, the heuristic has been modified to better identify good splits in the current state. The basic process works as follows. First, build the regression graph looking for good actions to apply to the current state. If an action is found, return it. However, if no action is found (in DAP and pDAP, an action can only be applied if all its preconditions are fully ground), then mark all literals that are abstractly specified in the current state (e.g., not fully ground) and re-calculate the paths as if they were true in the current state. The abstract literal with the shortest path to the goal is then selected as a

split candidate.

This revised regression graph heuristic produced mixed results. Some problems were solved very quickly, but on many problems, pDAP was still splitting on many irrelevant features. An analysis of the heuristic showed that early on in the search, when most of the literals are not ground in the current state, the heuristic was unable to decide which of a large set of literals with identical path lengths to the goal were relevant and which were not relevant. The following minor modifications to the scoring method addressed this problem.

Prefer goal literals — Goal literals that eventually need to be split upon at some time, so if there is no better option, split on a goal.

Prefer literals in initial states — If nothing else is better, try to reduce the difference between the current abstract state and the initial states.

Prefer literals that support many paths — If a literal is on more than one possible path to the goal, then give it greater preference, since it is most likely part of a solution.

The resulting heuristic is very effective in the robot domain, nearly eliminating irrelevant splits produced by pDAP. However, further analysis has shown that the modified regression graph heuristic is also very effective in guiding DAP, to the point where the additional overhead incurred by pDAP to justify each split, causes pDAP's runtimes to be significantly longer than DAP's.

The result of these modifications allows MASA-CIRCA to efficiently solve problems within Kabanza's [15] modified robot navigation and object-delivery domain and compare favorably with both Kabanza's and Pistore & Traverso's [33] results.

6. Probabilistic State Space Planner

A real-time agent with limited perceptual, computational and actuator resources must carefully allocate these resources at execution time to do the best job it can in monitoring for aspects of the world state and responding quickly to emerging hazards in its complex and dynamic environment. If the resource-limited agent allocates more of its resources (or some of its resources more frequently) to monitoring for some states (or state features), then it will be less capable of tracking others successfully. Similarly, it cannot be assumed to execute all the resource-demanding actions equally well. Designing an effective schedule of monitoring for and responding to the most important situations at the right times with the available resources is thus a complex optimization problem.

To make worst-case performance guarantees, an agent must ensure that it has set aside sufficient resources for monitoring and responding to the important situations. Those resources are only fully utilized, however, when the situations requiring remediation actually occur. In a sense, monitoring for and being prepared to react to situations that do not arise is, retrospectively, a waste of resources. When pushed past the limits of its

capabilities, a reasonable heuristic for an agent to shed some of its load is to remove the monitoring and response activities that are most likely to waste resources. In other words, all other things being equal (including the costs of failing to respond to various situations), an agent should preferentially monitor and respond to situations that are more likely to occur over those that are less likely to occur.

However, determining the likelihood of a real-time agent encountering a particular situation or a state can be challenging because the likelihood is dependent not only on the actions the agent should perform, but also on its choices of how frequently it checks whether to perform them. By definition, a dynamic environment is one in which the state can change via events outside the agent's control. In general, the sooner a real-time agent detects and responds to a situation, the less opportunity there is for the environmental dynamics to intervene, and the higher the chance that the agent is going to meet its deadlines. The probability of encountering a state thus depends on a complex temporal convolution between the agent's plan and the exogenous events.

One contribution of our work is that we have developed a probabilistic action and event model to efficiently approximate the transition probabilities for a real-time agent. We use this to augment the State Space Planner (SSP) to permit the estimation of probabilities of encountering situations. The resulting Probabilistic State Space Planner (PSSP) supports a second contribution, which is our strategy for using these state probabilities to prioritize resource expenditures when the agent cannot be prepared for all eventualities. Finally, a third contribution of our work has been to streamline the search through the alternative reachable state spaces by improving the backtracking and pruning techniques employed in the search [5].

6.1. Technical Advances

The details of our methods for modeling, computing, and using state probabilities in the PSSP are covered in more depth elsewhere [17], [19], [18]. Here we summarize them at a high level.

State transitions, whether caused by an agent's own actions or by external events, are modeled by representing the state features that must hold before the transition (the preconditions) and those that are changed after the transition (the effects). We augment the model to capture stochastic and temporal aspects by giving each transition model a probability function describing its likelihood of occurrence over time. Specifically, let T be the random variable denoting the time that a transition fires since it was enabled *f(t).* t is the transition time, ranging from 0 to infinity. To model the execution of a real-time agent as a continuous-time stochastic process, we assume that all transitions are mutually independent. After numbering the transitions in a state, we denote the i-th transition by tt_i. The transition probability of tt_i, relative to all other applicable temporal transitions in a state, is the probability that its firing time, T_i, is the minimum, T, among all transition times.

We represent the time-dependent probabilities with what we call probability rate functions.

For a (temporal) transition tt, a user specifies a probability rate function that specifies, for each time interval h since the transition has been enabled, the probability that tt fires in that h^{th} time interval, given that the transition has not fired before in any of the previous $h-1$ time intervals. For example, if a fair coin is flipped each second, the probability rate for the "heads-to-tails" transition is 0.5 over each second, regardless of how much time has elapsed, given that the state is still "heads" after the flips so far.

To compute transition probabilities, we approximate the "true" continuous probability (rate) functions of the transitions by piecewise constant probability rate functions. To calculate the transition probabilities for a set of concurrent events and actions that match a state s, we compute the dependent probability rate function for each transition enabled in the state in each time interval. A dependent probability rate function of a transition in state s can then be computed that describes the probabilistic temporal dynamics of a transition in that state when there are other concurrent applicable transitions. With transition probabilities computed in this way, we can compute the state probabilities of the states in a stochastic process of a real-time agent by computing the state probabilities of the states in the corresponding embedded Discrete Time Markov Chain using standard Markov techniques.

These capabilities for computing state probabilities are exploited in the overall controller synthesis process as follows. After an agent's Probabilistic State Space Planner generates a tentative plan, the plan is passed to the scheduler, which tries to schedule the actions according to the resource constraints of the execution platform. If all planned actions are schedulable, then it is done. Otherwise, the PSSP computes the state probabilities and the actions planned for the least likely states are removed in increasing order of their state probabilities until the remaining set of actions is schedulable.

The intuition is that if an agent has a very low probability of reaching a state, then ignoring the hazards or Temporal Transitions to Failure (TTFs) in this state does the least harm (assuming all failures are equally bad). We call this the unlikely state (cutoff) strategy. The agent's failure probability increases by no more than the probability of the state the action was planned for. Note that, if all failure states are not equally bad, our strategy can be modified to accommodate domains with varying degrees of failure. For instance, instead of ignoring the least likely states, we could ignore the least deleterious (disutility) states weighted by their state probabilities.

It could well be the case that the risk involved with ignoring the lower-probability states is more broadly unacceptable. It is for this reason that the PSSP is just one component of our larger CIRCA architecture. Specifically, the AMP is responsible for assessing the "big" picture, and determining whether to adopt the PSSP recommendations, or to reformulate the controller synthesis problem in some other way. The AMP and PSSP thus co-routine to negotiate on a synthesized controller that meets the mission needs as represented by the AMP, while at the same time meeting the execution platform constraints as represented by the CSM (the PSSP and the scheduler).

The above assumes that an agent generates an initial tentative plan. Planning involves a forward search through a potentially large state space, and the space can include various deadends as the planner discovers and has to avoid action choices that lead down paths along which failure cannot be avoided. In general, therefore, this can be a costly, exponential search. We have developed techniques for dependency-directed backtracking, and for pruning portions of the search space based on ideas from constraint satisfaction search, that lead to substantial planning speedups [5].

6.2. Evaluation and Demonstration

We have evaluated the quality and effectiveness of the PSSP [17]. First, we have shown analytically and demonstrated empirically that the piecewise linear approximation that we use correctly estimates probabilities for Markovian systems, where the estimates improve (and approach the true values) as the discretization becomes increasingly fine. For non-Markovian dynamics, we have developed a Monte Carlo simulation-based tool for estimating state probabilities. We have also coupled both approaches together, so that the more efficient methods can be used for most of a reachable state space but simulation can be employed for the subspaces that are non-Markovian.

We have demonstrated the efficacy of the PSSP in a UCAV domain. Specifically, we have looked at a problem in which the UCAV cannot monitor and respond to all types of missile threats. We have shown that a naive approach (corresponding to our earlier methods for dropping tasks in overconstrained cases) can make very poor choices because it only considers the likelihood of a transition without considering the likelihood of reaching a state where that transition can even occur. Our new methods do consider state likelihood, and thus generate schedulable control plans that minimize the risk faced by the resource-limited UCAV.

The University of Michigan and Honeywell have integrated the PSSP capabilities into the overall architecture by defining an API through which the AMP and the CSM communicate. This permits the swapping of different CSM modules into the architecture. Using this interface, we have implemented and demonstrated a simple negotiation process between the AMP and the CSM. In this process, the AMP provides a controller synthesis specification to the CSM, which in turn attempts to formulate a schedulable controller that meets the specifications. Because the problem overtaxes the execution platform, the CSM fails, but using the PSSP it discovers a probability threshold such that, if it can ignore threats in states that are unlikely to be reached (whose probabilities of being reached are below the threshold), then a schedulable controller is likely to be formulated. The AMP receives this information, and can decide to update its controller synthesis specification and repeat the request.

Furthermore, we have also demonstrated how this negotiated interaction between the AMP and CSM can serve to support time-critical planning activities. Specifically, if the AMP is faced with synthesizing a controller quickly, it can provide the PSSP with a specification with a relatively high probability threshold. The PSSP can use this to quickly generate a

controller that only includes actions for the most likely threats to be realized (for states whose probabilities meet or exceed the threshold). The AMP can iteratively reduce the threshold, in an "anytime" manner, until either a sufficiently safe controller is formulated or until the deadline for adopting a controller is reached.

Finally, the improved backtracking and pruning techniques have been incorporated into the planner, and evaluated over a set of a few thousand randomly-generated cases. The improvements led to some planning speedups of over an order of magnitude [5].

7. Resource-Driven Subgoaling

When the demands placed on a controller to be synthesized outstrip the capabilities of the execution platform, something must be sacrificed. Section 6 describes a strategy for ignoring the least-likely threats to arise so as to concentrate on guaranteeing actions to preempt the most likely threats. An alternative to making such probabilistic guarantees is to instead make temporal guarantees: that failures cannot be reached from states that will be reached soonest, or before some particular action or event occurs. That is, rather than guaranteeing safety indefinitely from the most likely threats, the system could guarantee safety in the near term from all threats.

Resource-driven subgoaling attempts to group states together based on when they can occur, with particular emphasis on being able to control transitions from one group of states to another. Specifically, the goal is to analyze the graph of reachable states and transitions among them, and break that graph into a series of subgraphs that correspond to different "phases" of the overall mission. Transitions from one phase to another should be under the control of the agent. For example, an overall mission of getting a jet from a gate in Detroit to a gate in Minneapolis could require a very complicated controller that could not be implemented. However, breaking the mission down into phases (such as taxiing from the gate to the runway, taking off, cruising, landing, and taxiing) might succeed since the set of actions and threats in each phase is more manageable.

7.1. Technical Advances

We have developed algorithms that decompose a mission, represented by a state graph with an initial state and a goal state, into phases which are separated by future actions that the CIRCA system can perform. Each phase consists of an initial state and one or more subgoal states. Because a phase generally involves fewer states, hence TTFs, hence actions to schedule to preempt TTFs, CIRCA can make more stringent guarantees about failure avoidance within a phase than it can if it must plan for the whole mission at once.

Our algorithms mark the states within the state graph such that states that can be reached through temporal transitions or through actions that preempt temporal transitions are marked as being within the same mission phase (what we refer to as being part of the same mega-node). Mega-nodes are separated by action transitions that have no timing constraint on them. The graph-coloring process by which states are marked (mega-nodes are generated) is polynomial in the number of reachable states. See [34] for details of the

algorithm and examples of its execution.

7.2. Evaluation

We have conducted a number of experiments to study our algorithm. First, we have looked at whether it can process relatively large state spaces to find mission phases relatively quickly. Our most recent implementation of the algorithm can process a state space of 1000 states in less than one second by exploiting polynomial-time graph-searching algorithms.

Second, we have investigated the conditions under which finding mission phases is likely to be successful. The premise of our algorithm is that various clusters of states are disconnected from other clusters except through a single thread corresponding to an agent's volitional action. Not surprisingly, our empirical analyses confirm that a greater number of non-volitional transitions impede the discovery of useful phases, since they make all states reachable beyond the control of the agent. Similarly, as the number of transitions to failure rises, the number of actions that an agent *must* do grows, leaving fewer opportunities for volitional movement between phases.

Using the defined API for communication between the AMP and CSM, the resource-driven subgoaling algorithm generates output that can be supplied back to the AMP. The output is essentially a list of phases, represented as smaller controller-synthesis problems specifying initial and goal states corresponding to the ways in which each phase could be entered and then left for a subsequent phase. Given this information, the AMP can feed to the CSM portions of the mission a phase at a time to permit the overall mission accomplishment with higher guarantees of success, since each phase is more likely to be fully schedulable.

8. Constrained Markov Decision Processes

In a number of respects, the problem that CIRCA faces in synthesizing a controller is similar to the creation of a control policy in Markov Decision Processes (MDPs). Besides the fact that the underlying processes about which CIRCA must reason are often non-Markovian, a second significant difference is that traditional MDPs are not concerned with resource-limitations of the policy execution platform. In CIRCA, such limitations are of central concern. Thus, while a traditional MDP can focus on developing an optimal policy by combining optimal action choices for different situations, this cannot happen with CIRCA because the resources devoted to optimizing some part of the policy mean that some other part of the policy will lack sufficient resources to be optimally addressed.

Yet, the principled underpinnings of MDPs, the explicit representation of expected utilities, and the tools for optimization, present an attractive counterpoint to the generally heuristic methods used by CIRCA to derive a satisfactory controller with no assurances about optimality. We have begun an investigation into whether MDP techniques can be adapted for CIRCA needs, and what costs and benefits will result in doing so.

8.1. Technical Advances

We have developed a characterization of a variety of types of resource-limited constraints that could be placed on the policy search process, including constraints on executing a policy (such as limits on the power available to a robot carrying out a policy) and constraints on operationalizing a policy (such as limits on the size or complexity of a policy that an agent can effectively use to map states to actions). Conceptually, the latter resembles the execution resource limits respected by CIRCA, in which only a limited set of periodic TAPs can be scheduled to meet their timing requirements.

Some types of constraints can be dealt with by extending prior work on Constrained MDPs (CMDPs). CMDPs develop policies which, in the expected case, will not overconsume particular resource costs. However, in keeping with CIRCA's traditional concerns about catastrophic failure, we have studied the case where overconsumption should be avoided much more emphatically than just in the "expected" case. For example, if the policy is for a UCAV, there is definitely a significant cost in being in midair when the fuel runs out.

We have developed techniques that augment past linear programming algorithms for CMDPs to incorporate constraints that reflect a user-provided probability of overconsuming a resource [6]. The lower the probability that a user wants to tolerate, typically the more conservative the resulting policy is. In the extreme, an UCAV might choose not to take off at all to avoid any risk of running out of fuel while airborne! More generally, our methods support trading off utility (goal achievement) for safety (avoiding overconsumption).

We have also developed techniques involving mixed integer programming to work with constraints on the kinds of policies that can be operationalized. For example, some architectures might only support deterministic policies (that map each state to a single action, rather than permitting some weighted randomization among actions). While a randomized policy might be optimal, mixed integer programming techniques can be used to find the optimal deterministic policy in an efficient manner [7],[8].

8.2. Evaluation

We have run experiments over a number of random problems (50 problems per data point) where for each data point we averaged the probability of resource overconsumption and the utility received, as we increase the allowable probability of overconsuming a resource. The results (see [6]) confirm that our techniques do ensure that the risk of resource overconsumption is no greater than the user-specified bounds. In fact, often it was much lower. Because utility is generally positively correlated with resource consumption, it is not surprising that utility (when resources are not overconsumed) is lower for our more conservative method than traditional methods for CMDPs and unconstrained MDPs. We are currently seeking means for establishing constraints that still ensure that overconsumption occurs below the specified probability, but not so far below as our current methods so that we can accrue more utility.

More broadly, we have begun to put together analytical results across a spectrum of

different types of constrained MDP problems. In particular, we have been able to show that, while the execution constraints studied above permit algorithms that are P-complete, other constrained problems appear to be NP-complete, including problems of being constrained to deterministic policies, and of being constrained in the total utilization of actions in a policy [8].

9. Communication to Reduce Uncertainty

As was mentioned previously (Section 6), an agent wastes precious resources of its execution platform if it includes in its synthesized controller monitoring and response actions for situations that do not arise. Clearly, if an agent can establish that a situation *cannot* arise, then it can safely remove the corresponding reaction to that situation from its control plan.

One way in which an agent can do this is through communication with other agents [9]. An agent might prepare for all states it may reach as a result of not only its own actions and the environment transitions, but also as a result of the possible actions that other agents are capable of taking. Just because an agent is capable of taking an action, however, does not mean that it will take that action; anticipating all possible actions on the part of other agents requires an agent to prepare for states that might never arise.

It can therefore be to an agent's advantage to engage in a dialogue with other agents to find out which of their potential actions in a situation they have actually decided that they will do. That is, to learn about aspects of their synthesized controller. Armed with such knowledge, an agent can begin to prune away portions of its state space, and hence prune actions that it feared it would need to take to preempt potential failures.

9.1. Technical Advances

We have developed a protocol, which we call the Convergence Protocol, that enables agents to prune (ignore) states that others can tell them are unreachable, and thus for which reactions need not be planned and scheduled. Details of the protocol and its evaluation are presented in [20, 21].

Using this protocol, an agent first determines whether it cannot schedule all of the TAPs that it considers vital for safety. If it can schedule them all, then it need not initiate any further communication. However, if it is indeed overconstrained, then the agent will inspect its reachable state graph and find the states from which some other agent appears to have a choice of actions. It picks one of these, and inquires from the corresponding agent about the action that the agent has decided it will do if the state is reached. Upon receiving this information back, the agent can essentially mark the transition probabilities of the other actions as zero, and remove from its state graph any states that are thus rendered reachable with a probability of zero! Actions associated with those states are also removed, and the scheduling problem for the remaining actions might be made easier.

The agent repeats this process until it can schedule all of its remaining actions, or until it

has no more states about which it is uncertain of others intentions. In that latter case, it can resort to the use of state probabilities to decide which actions to drop, as in Section 6.

How an agent decides which state to inquire about next depends on which heuristics it adopts. It could choose randomly, but more informed heuristics attempt a greedy strategy of trying to ask about states whose answers will most effectively trim actions from the necessary set of actions. We have developed and experimented with several candidate heuristics.

In addition, if after completing the use of the protocol the agents still cannot guarantee all of their actions, an alternative to using the probabilistic methods and incurring consequent risk is to instead consider alternative combinations of actions, either for themselves or others. This provides an opportunity for not only communicating to inform others, but also communicating to negotiate with others to get them to change their plans. However, the space of possible joint plans is combinatorial in size, and so a thorough exploration of this space is typically infeasible. We have been developing tractable negotiation methods that employ hill-climbing techniques for exploring a promising portion of this space.

9.2. Evaluation and Demonstration

To evaluate the merit of the Convergence Protocol, we have generated a set of random domains. Each domain has a random number of agents from 2 up to a maximum of 10. Each agent has its own knowledge base. The knowledge base has 7 private and public binary features (T/F) total. The number of public features in a domain is random. It measures how tightly coupled the agents are for that domain, i.e., how many features they have in common. As any public feature is shared by all agents, the knowledge bases for any given domain have the same number of public features. There are 15 private and public actions combined, and 7 private and public temporal transitions combined for each agent. The compositions are random. The actions and temporal transitions are generated such that they invert the values of a random number of features. All knowledge bases for a domain contain the same set of public temporal transitions and public actions.

We have generated 1626 agents for 402 domains with which we perform our experiments. The number of agents that are able to schedule for all their TAPs before running the Convergence Protocol is 202 (12.42%). The number of agents that are able to schedule for all their TAPs after running the protocol is 704 (43.30%). In other words, 502 (30.87%) agents become able to schedule for all their TAPs. For those agents that still fail to schedule for all actions, they, nonetheless, drop fewer necessary actions (by raising the probability threshold) than they otherwise would have needed to. For our experiments, the reduction in the number of necessary actions dropped is on average 59.55% with a standard deviation 39.85%. As a result, the agents' utilities are not as compromised as they would otherwise be without running the Convergence Protocol.

We have also conducted a number of other experiments and analyses [21]. We have also investigated the features of a set of CIRCA knowledge bases, and of reachable state graphs, that correlate to effective employment of the Protocol.

Finally, to demonstrate and illustrate the protocol, we have generated a series of video clips that show how an agent builds its own reachable state space, and then engages in the protocol to trim from this space the states that turn out to be unreachable based on the action choices of the other agent. The video clips are available at `../UMDemo2003`, along with a set of Powerpoint slides that explain the demonstrations.

10. The Adaptive Mission Planner (AMP)

In the preceding SA-CIRCA project, we developed requirements for the AMP and investigated two existing planners, SIPE-2 and Prodigy, as possible starting points. Neither provided the flexibility and power required, so we developed an entirely new AMP in this project. This has turned out to be a useful and compact module that was developed with relatively low cost. In this section, we provide an overview of the previously-defined requirements and describe the new AMP design. In addition to the material below, the AMP and the underlying deliberation scheduling algorithms are discussed in detail in several publications [31, 11, 23, 25, 28].

10.1. Requirements for the AMP

Based on prior experience with the CIRCA architecture and our scenario designs for the UAV demonstration domain, we developed a set of general functional requirements for the AMP. We want the AMP to:

Decompose a Mission into Phases — The AMP must divide up the overall mission (and associated problem state space) into a series of mission phases with overlapping state space regions.

Create CSM Problem Configurations for the Phases — For each mission phase, the AMP must build a problem configuration that it sends to the CSM for planning. The CSM will builds real-time reactive control plans for each of these problem configurations.

Modify Configurations when the CSM Fails — If the CSM fails to generate a safe controller for a given subproblem, the AMP must modify its plans to reduce the complexity of that phase.

Set up Execution-Time Sentinels/Monitors — The AMP must be able to automatically monitor its own performance to detect deviations from the plan.

Incorporate Changes During Execution — Because the AMP is the long-term planner in CIRCA, it must be able to handle deviations from its coarse plan as the situation progresses.

Manage Reasoning Resources (Deliberation Scheduling) — The AMP must control the CIRCA reasoning process itself, so that the CSM builds controllers in a timely fashion; if the phase problems sent to the CSM are badly formed, the CSM may never return success or failure. The AMP must monitor CSM performance and adjust its plans to meet the soft real-time deadlines imposed by the domain.

```
(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 10. Simplified Lisp code for the AMP outer loop, processing tasks and messages.

Plan for Multiple Agents — Distributed applications of CIRCA will require the AMP to communicate and coordinate with other agents to effectively manage roles, responsibilities, and closely-coordinated behaviors.

Direct Execution — The AMP should direct the overall execution of real-time control plans by managing the plan cache in the RTS; this requires interleaving execution and planning.

Search — The system must maintain an explicit representation of planning decisions for possible re-evaluation and change.

In the original SA-CIRCA formulation of these requirements, we were focused on making the AMP a projective planner that could reason about abstract phase plans and project such aspects as fuel consumption. In the MASA-CIRCA ANTS-funded project, we re-focused our requirements to emphasize the AMP's deliberation scheduling (CSM control) and multi-agent negotiation functions. Our new design meets all of the above requirements.

10.2. AMP Design

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 10 shows a simplified version of the Lisp code used to implement this loop. The real system is somewhat more complex because it remains responsive to incoming messages even while executing some tasks, particularly while running CSM planning tasks that can take significant time.

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 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).

11. Deliberation Scheduling

For CIRCA, deliberation scheduling is the task of deciding what problems the AMP and CSM modules should be working on at any time. Most importantly, the time-consuming planning and scheduling processes within the CSM must be managed to ensure that the best possible control plans are built throughout the mission. Our first paper in the Self-Adaptive Software workshop series describes ways in which different CSM plans can alter system performance [23, 25].

After considering a number of heuristic strategies for deliberation scheduling (e.g., earliest-first or shortest-first), we settled on pursuing a more principled decision-theoretic approach. We modeled the deliberation scheduling problem as a Markov Decision Problem, so that an optimized policy for the MDP can be used to control the CSM processing (deliberation) to yield the maximum expected utility. By focusing on maximizing the expected utility of deliberation, this approach can smoothly balance the competing pressures of time pressure (tending towards shortest-first planning), self-preservation (tending towards threat-first planning), and goal achievement (tending towards goal-first planning).

In [11] we describe this approach and address the time cost of the meta-level decision itself, by developing computationally feasible heuristics that make deliberation scheduling decisions "greedily" but quickly. To assess the performance of these greedy heuristics, we

describe simplified Markov Decision Process (MDP) models of the AMP's deliberation scheduling problem and assessing both the optimal and greedy solution policies. Our results indicate that these greedy heuristics are able to make high-quality deliberation scheduling decisions in polynomial time, with expected utility measures quite close to the NP-complete optimal solutions.

Finally, we implemented the deliberation scheduling strategies developed and evaluated in [11] into our evolving AMP prototype. Our most recent Self-Adaptive Software workshop paper [28] describes this implementation and how it meets the objectives for the AMP, and describes an example scenario comparing several different deliberation scheduling algorithms. Section 13.2 includes that demonstration description, and also shows how the AMP's decision-theoretic deliberation scheduling was critical to MASA-CIRCA's ability to respond smoothly and intelligently to unexpected situations, as described in Section 13.4.

12. Demonstration Environment

The MASA-CIRCA advances made in this project are all domain-independent, and can be applied to a wide variety of real-time agent control problems. Most of the new capabilities have been demonstrated in the context of simulated unmanned air vehicle combat missions. Illustrative demonstrations were produced using an existing CIRCA-compatible flight simulation to provide simple combat-oriented simulation functions.

12.1. The CIRCA Air Combat Maneuver Simulation

Leveraging prior work on flying UAVs with CIRCA (both at Michigan and Honeywell), we began with a demonstration system capable of simulating a simple F-16 aircraft model flying over high-resolution terrain. The CIRCA controller was interfaced at a fairly high level, providing waypoints and other discrete commands (e.g., deploy flaps) to the simulated aircraft. The simulation itself provided a simple autopilot to fly towards waypoints, and an auto-throttle function to maintain appropriate speeds.

The simulation provides medium-fidelity combat simulation features including:

- Deployment of chaff and flares.
- Automated evasive maneuvers.
- Ground-based threat installations (SAM sites and IR launch sites).
- IR-guided and radar-guided missiles that track the aircraft and are distracted by flares or chaff, respectively.
- Air-launched missiles that track ground targets.
- Smoke trails for aircraft and missiles, to provide persistent trajectory display.
- Projected future waypoint/path display.

Figure 11 shows a screengrab from one simulation run, in which the UAV has been threatened by a surface-to-air missile (seen as a black line curving towards the aircraft) and it is responding with flares (one of which is seen falling below the flight path) and a counter-strike missile (seen arcing down onto the surface installation).

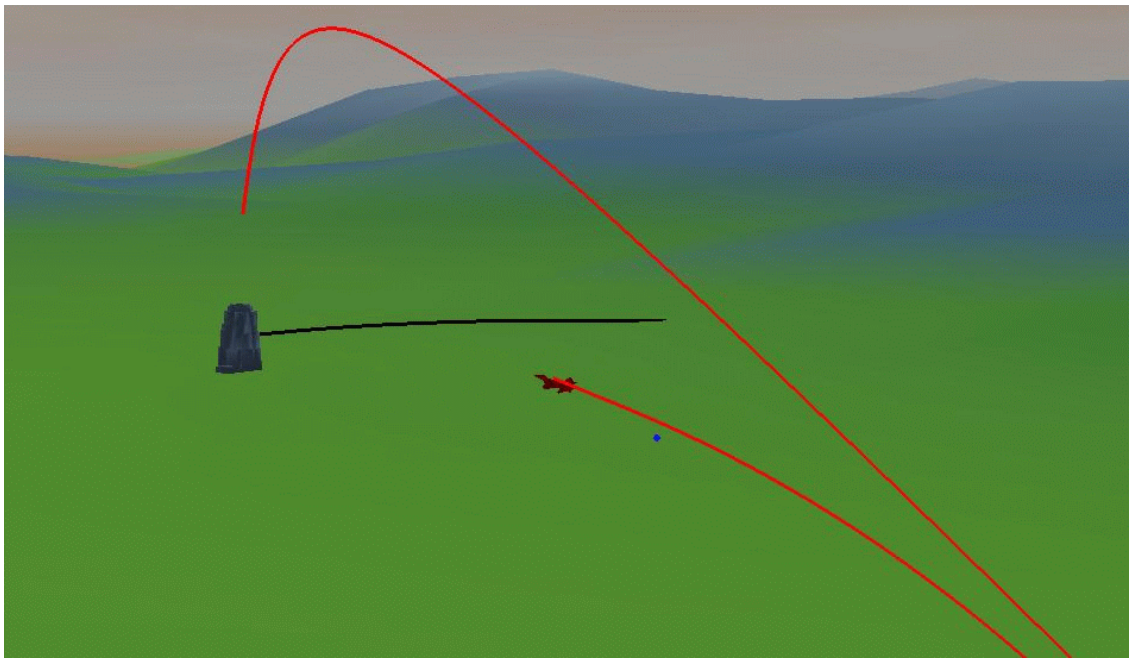


Figure 11. The demonstration simulation illustrates a MASA-CIRCA-controlled aircraft responding to attacks with evasive maneuvers, flares, chaff, and counterattacks.

12.2. AMP Information Display

To provide a compact graphical indication of what functions the AMP is performing, we have developed an AMP Information Display (AID). The AMP sends its status information to the AID over a socket. Illustrated in Figure 12, the AID gives the observer visibility into the multi-agent negotiation and planning processes.

At the top of the AID, two labels indicate which CIRCA agent the display is describing, and what mission phase that agent is currently executing. Below those lines, two light bars labeled "IN COMM" and "OUT COMM" flash when the agent is receiving or sending over its socket connections to other agents. Further down, the aircraft icon will display the status of various aircraft subsystems. This aspect of the AID is obviously domain-dependent, and it is not fully implemented yet. In the future, we plan to add iconic representations of engines, defensive systems, and other subsystems subject to damage and imperfect operation. Currently, the only functioning element of the iconic display is the center diamond, which acts as a "heartbeat" for the AMP, blinking periodically when the AMP is idle but still functioning.

Below the iconic display, a set of lines display the status of each of the contracts in the overall multi-agent system, according to the single CIRCA agent's view. Each contract represents a threat or goal that must be handled in a particular mission phase. Each line has a text label identifying the contract it represents, and a color (not apparent in

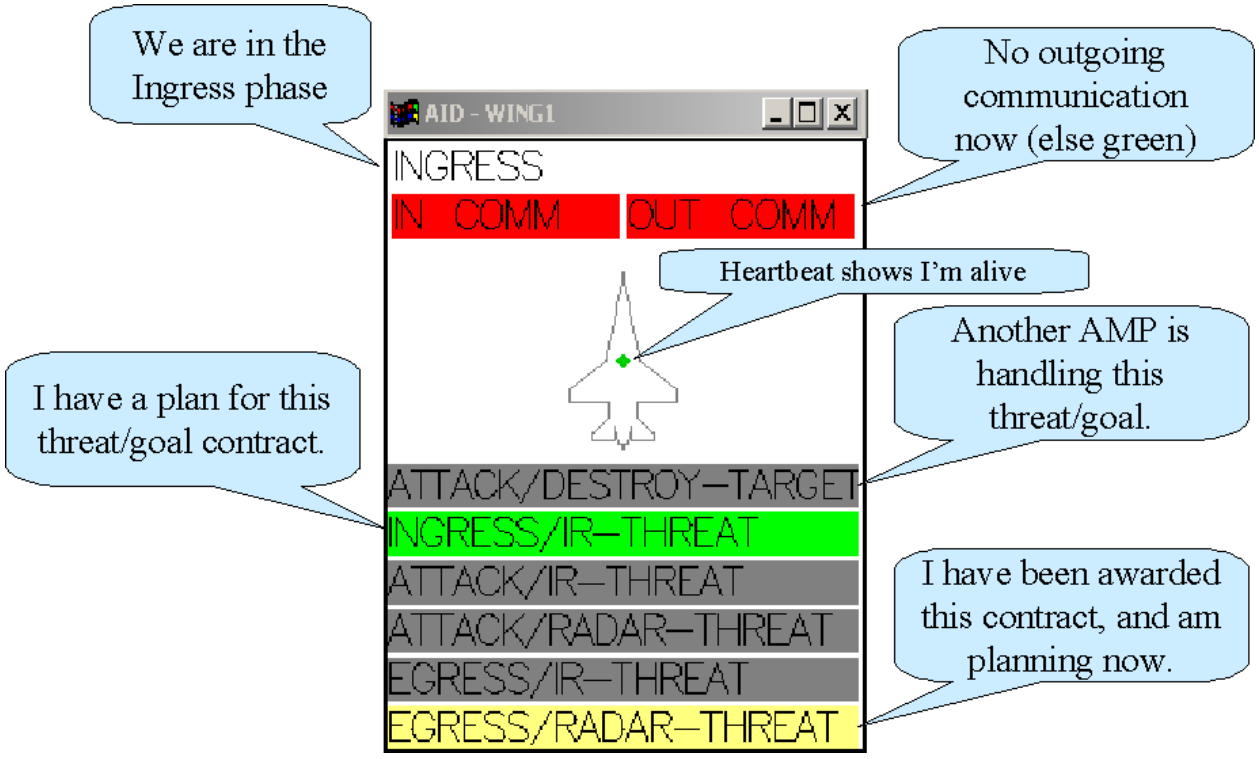


Figure 12. The AMP Information Display depicts AMP status.

black-and-white printouts) that describes the status of the contract:

Red indicates that a new threat/goal contract has arrived and is not yet even announced for bids by other agents.

Pink indicates that a new contract has been announced for bids, but all the bids have not yet been received.

Yellow indicates that this agent has been awarded this contract (i.e., it has accepted responsibility to "handle" this goal or threat).

Blue indicates that this agent's CSM is working on a plan to handle this goal or threat.

Green indicates that this agent has successfully generated a new controller (plan) to handle this contract.

Aqua indicates that this agent has successfully generated a plan to handle this goal or threat and is also working on a plan to handle it in combination with one or more other goals/threats (also colored green or aqua).

Gray indicates that another agent has responsibility for this contract.

Purple indicates that this agent was previously responsible for this contract, but is now disabled and will not handle the associated threat/goal.

The AID can also dynamically visualize the AMP's tradeoffs between deliberation time and the expected performance of synthesized controllers. Meters on the right side of the AID display the survival probability for different mission phases (using the current best controllers/plans) and the expected reward in each phase.

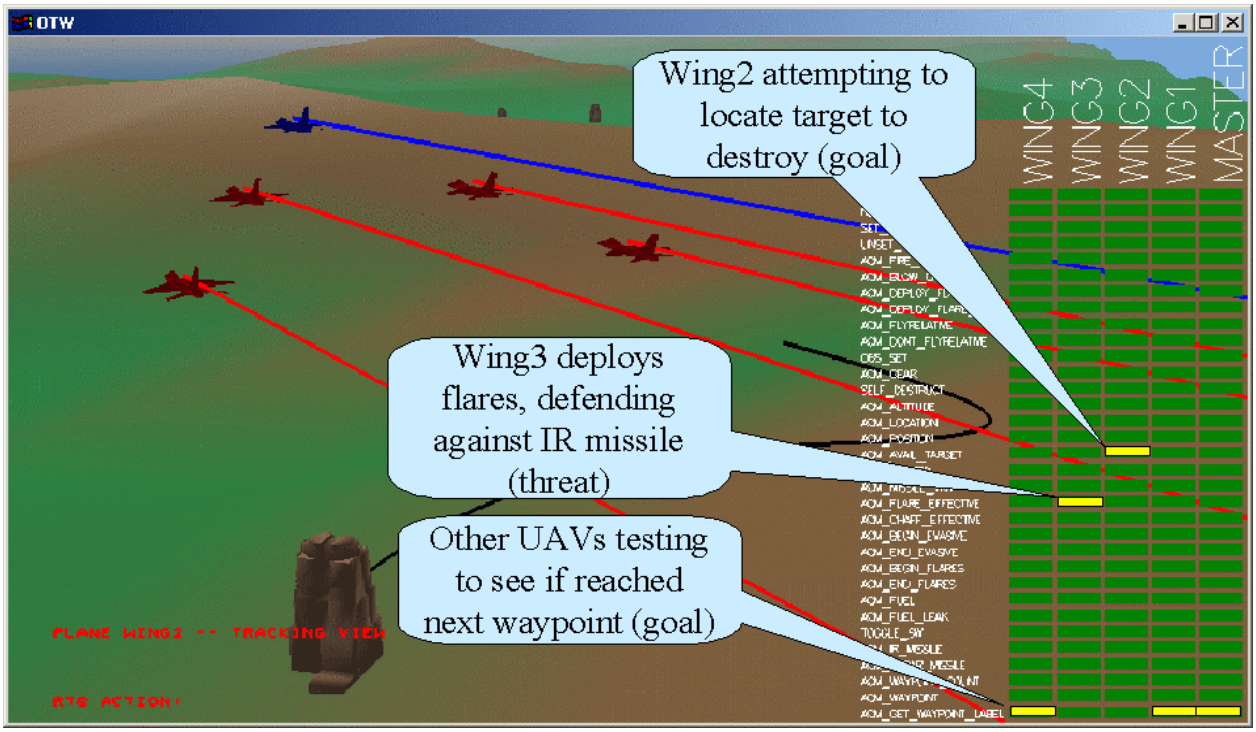


Figure 13. The RTS Information Display shows what each RTS is doing at any time.

12.3. RTS Information Display

The RTS can execute TAPs at extremely high speeds, so it is very difficult for an observer to follow its runtime activities. However, it can be very useful to have *some* indications of what the RTS is doing. To that end, we have developed an RTS Information Display (RID). As illustrated in Figure 13, the RID uses flashing bars of color to indicate each primitive test or action the RTS executes. As these color bars flicker rapidly on and off, the observer can recognize different patterns of RTS activity, and can also discern individual distinct actions that are infrequent. If a pattern of RTS activity persists for a few seconds, the observer can also identify specific activities the RTS is performing, by mapping the flashing bars back to the labels on the left edge of the display.

Recall that the AMP sends its status information to the AID over a socket. We could have designed the RTS/RID interactions in a similar way. However, because that socket communication could add significant overhead to the RTS operations, we have chosen instead to build this version of the RID directly into the flight simulation system used for our demonstration. This avoids both additional communication for the RTS and additional screen real-estate: the RID is drawn as a stencil over the top of the simulation display.

This approach has one notable weakness: the RID can only display RTS tests and actions that are sent to the simulator (i.e., those that interact with the world outside of CIRCA); it cannot show the RTS activities that are purely internal. For example, several TAPs in every TAP schedule are dedicated to downloading new TAP schedules, switching between TAP schedules when a suitable state is reached, etc. These "internal" TAPs cannot be

visualized on the RID embedded inside the simulator.

For situations where visualizing these internal activities is also important, we have a different version of the RID that runs in a standalone mode separately from the simulation environment. This version is currently only compatible with Xwindows displays. This version can show all RTS primitives calls, including internal functions such as reading in new TAP schedules (controllers).

13. Demonstrations

13.1. Demo 1: Team Coordination

13.1.1. Goal

The primary goal of this demonstration scenario is to illustrate negotiation between CIRCA agents and dynamic generation of controllers on the fly. The demonstration shows the agents negotiating diverse responsibilities and synthesizing customized controllers at the start of the mission, as well as re-negotiating roles and dynamically building new controllers during the mission when unexpected circumstances arise. A recorded movie of the simulation is available at `../movies/5plane-complete.mov`.

13.1.2. Mission

The demonstration scenario contains a flight of five unmanned F16-type fighter aircraft (`MASTER` and `WING1` through `WING4`) whose mission is to destroy a ground target while defending themselves against attack. Figure 14 shows a top-down view of the planned mission flight path, along with threat and goal location icons. The mission consists of three phases which correspond to three distinguished flight path segments: ingress, attack, and egress. During the mission, the AMPs need to create plans that account for several goals and expected threats:

1. Defending against IR-guided missile threats during the ingress phase.
2. Defending against IR-guided and radar-guided missile threats during the attack phase.
3. Destroying the target during the attack phase.
4. Defending against IR-guided missile threats during the egress phase.

In addition, one unexpected threat is displayed in Figure 14: a radar-guided missile site along the ingress path that is not reported to the CIRCA agents (presumably because it is unknown to our forces). This unexpected threat will turn out to be fatal to `WING4`, and this leads to the re-negotiation and replanning activity.

13.1.3. Demonstration Storyboard

When the mission begins, the fighters negotiate responsibilities for the first time. The `MASTER` is tasked with defending against radar-guided SAM sites during the attack phase of the mission. `WING1` gets this responsibility during the egress phase. IR threats are handled by `WING1` during ingress, `WING3` during attack, and `WING4` during egress. Responsibility for

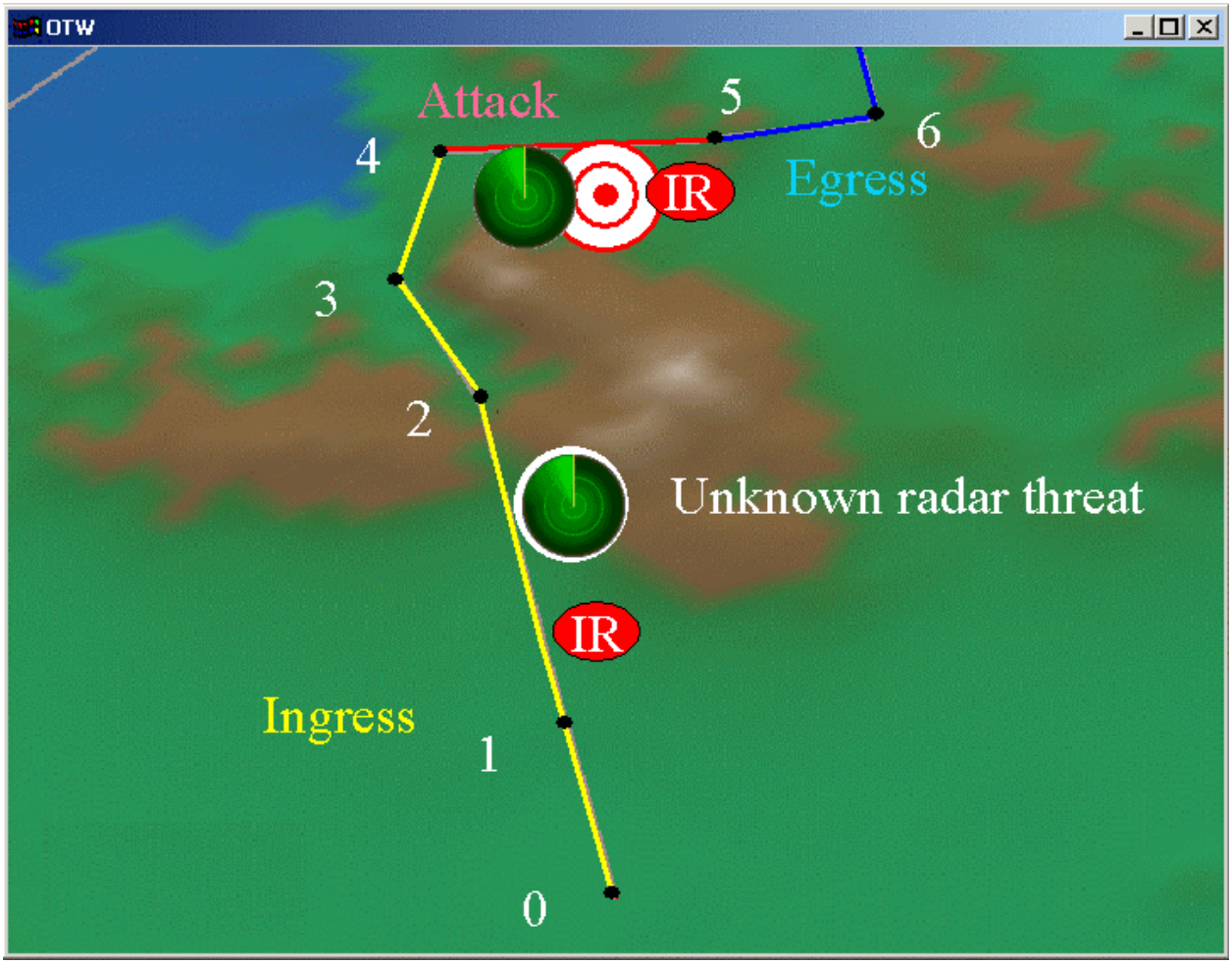


Figure 14. This mission overlay shows the expected path with known and unknown threats and targets.

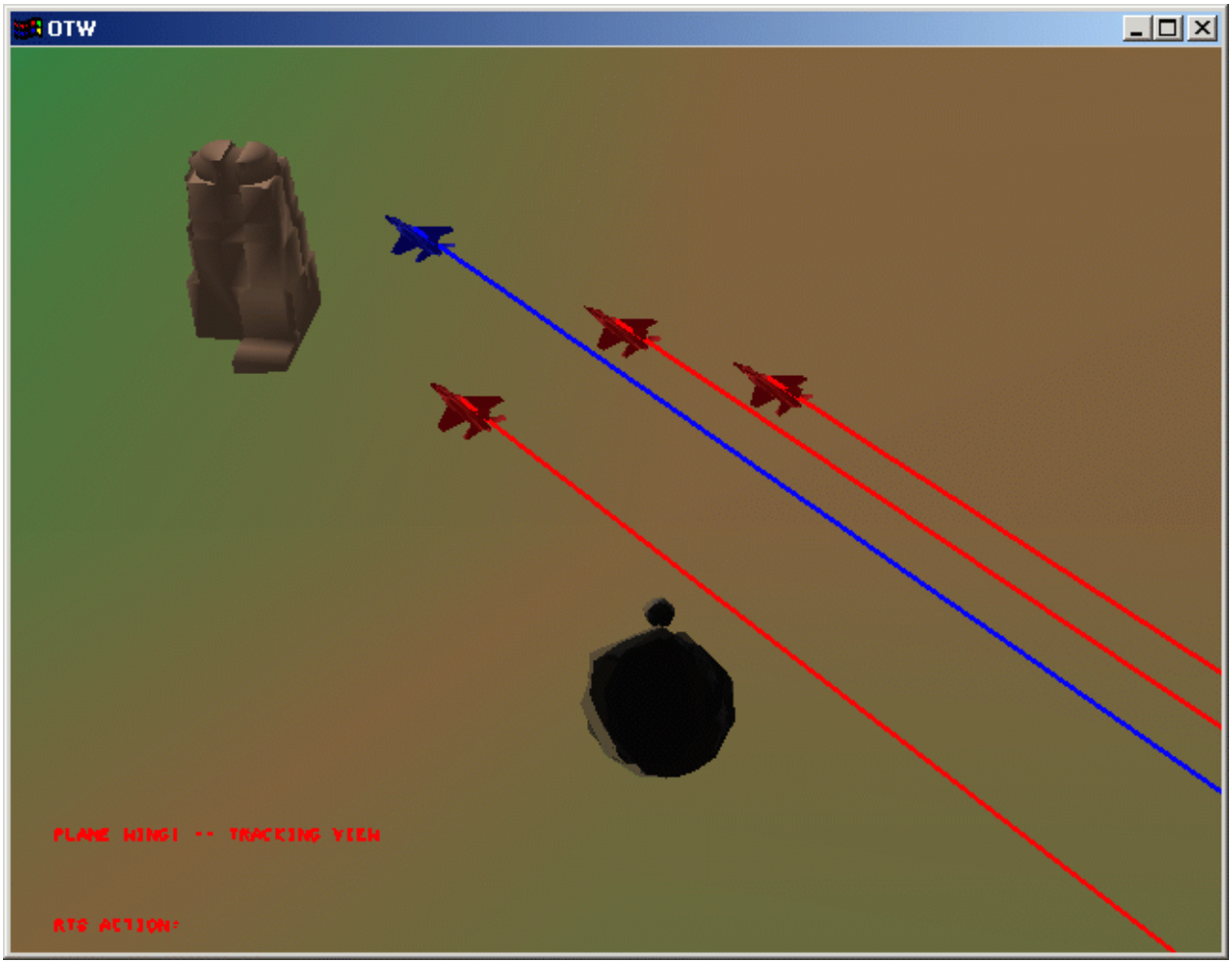


Figure 15. `WING4` exploding.

destroying the target is given to `WING4`. Once plans are constructed for the initial phases, the aircraft begin flying the mission (even before controllers for the latter phases are complete).

Before the aircraft leave the runway, controllers have been generated for all of the mission phases and the AID shows green lights for all of the goal and threat contracts. Then, shortly after the aircraft pass over their first waypoint, the anticipated IR threat attacks. Figure 13 illustrates the scene where `WING1` has been watching for IR threats and has begun deploying flares to confuse the incoming IR-guided missile. By repeatedly launching flares until the missile explodes on one, `WING1` successfully defeats the IR-guided missile.

A short time later the aircraft pass near the unexpected radar-guided missile site, which attacks. Unfortunately, since we did not tell the CIRCA agents about this potential threat, they have not built controllers that look for this danger. Unawares and unresponsive, `WING4` is destroyed (see Figure 15).

The AMPs immediately detect that `WING4` is dead, and re-negotiate its contract responsibilities. `WING2` gets the contract for destroying the target and `WING3` gets the egress

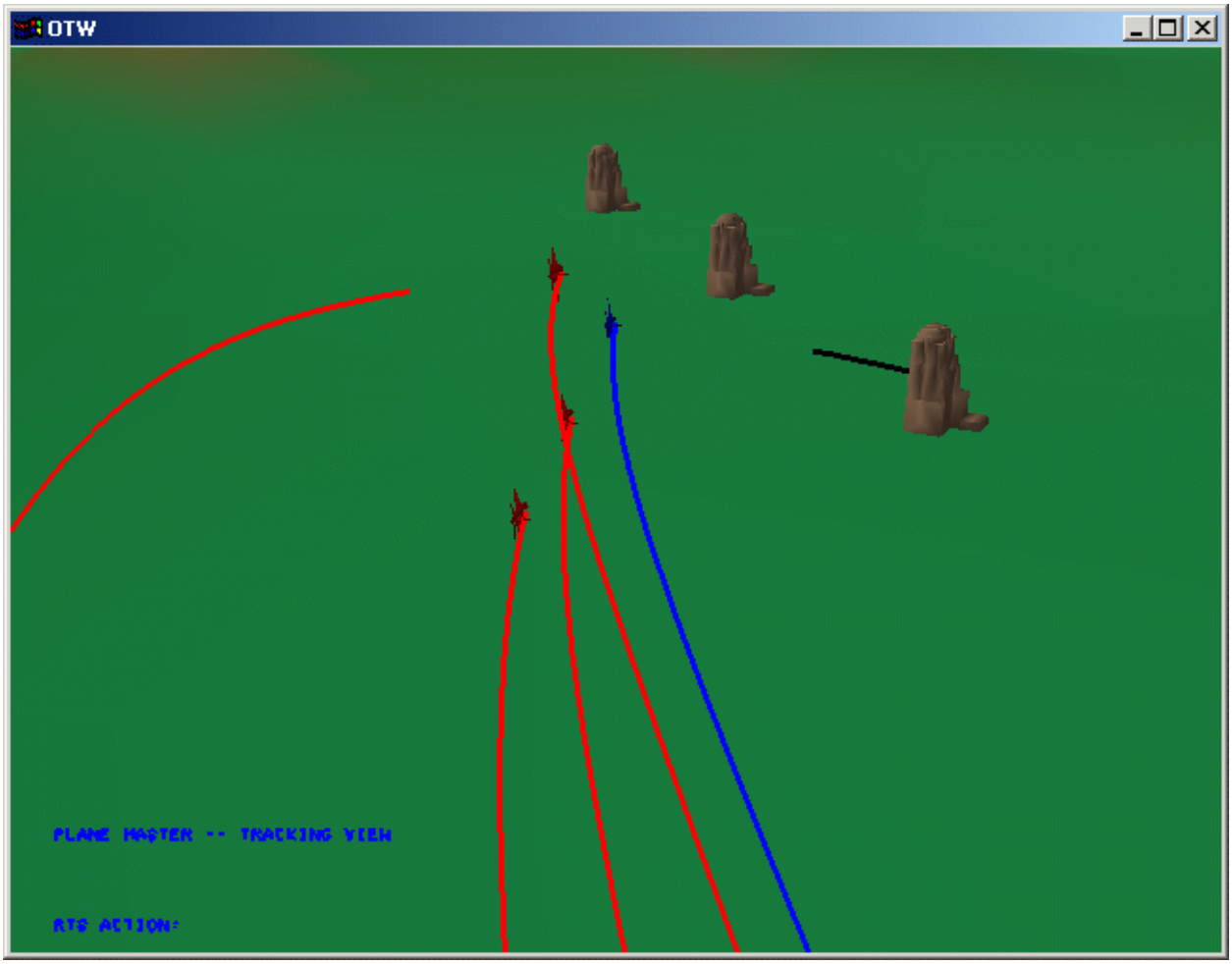


Figure 16. After `WING4` dies, the surviving CIRCA agents re-negotiate and generate new plans to handle its responsibilities and ensure mission success. Moments later, as the attack missile arcs in, the CIRCA team takes evasive maneuvers to avoid a rising radar-guided missile.

IR threat. Within three seconds the re-negotiation process *and* the new CSM invocations are complete, and the agents have created new controllers to handle their altered responsibilities. With all contract lights green (as shown in Figure 16), the team has recovered and the mission is still headed for success.

The aircraft continue to fly along the ingress flight route to waypoint four, where they enter the attack phase. When the target is in range, `WING2` fires a surface-to-ground missile. While that attack missile is on its way to the target, the fighters are threatened again (as illustrated in Figure 16). This time, the `MASTER` leads the flight into evasive maneuvers that defeat the radar-guided SAM. Figure 16 shows the aircraft at this time. During the evasive maneuvers, the second SAM site near the target launches an IR-guided missile at the team. In response, `WING3` deploys flares to defeat that missile. The aircraft then proceed safely to

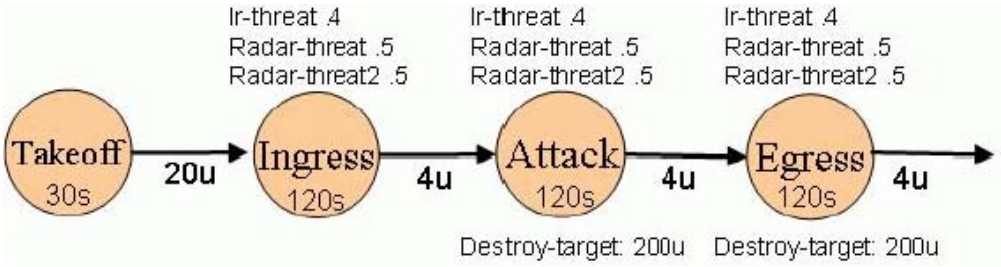


Figure 17. Each phase of the mission involves different threats and goals.

waypoint five. At waypoint five, the flight enters the egress phase of flight, the CIRCA agents begin executing a different set of controllers that are concerned about a different set of threats and goals, and the scenario proceeds without further incident.

13.2. Demo 2: Deliberation Scheduling

13.2.1. 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[4]. 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. A recorded movie of the simulation is available at ../movies/3plane-delib.mov.

Figure 17 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,

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

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 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 17 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.

13.2.2. 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 18 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

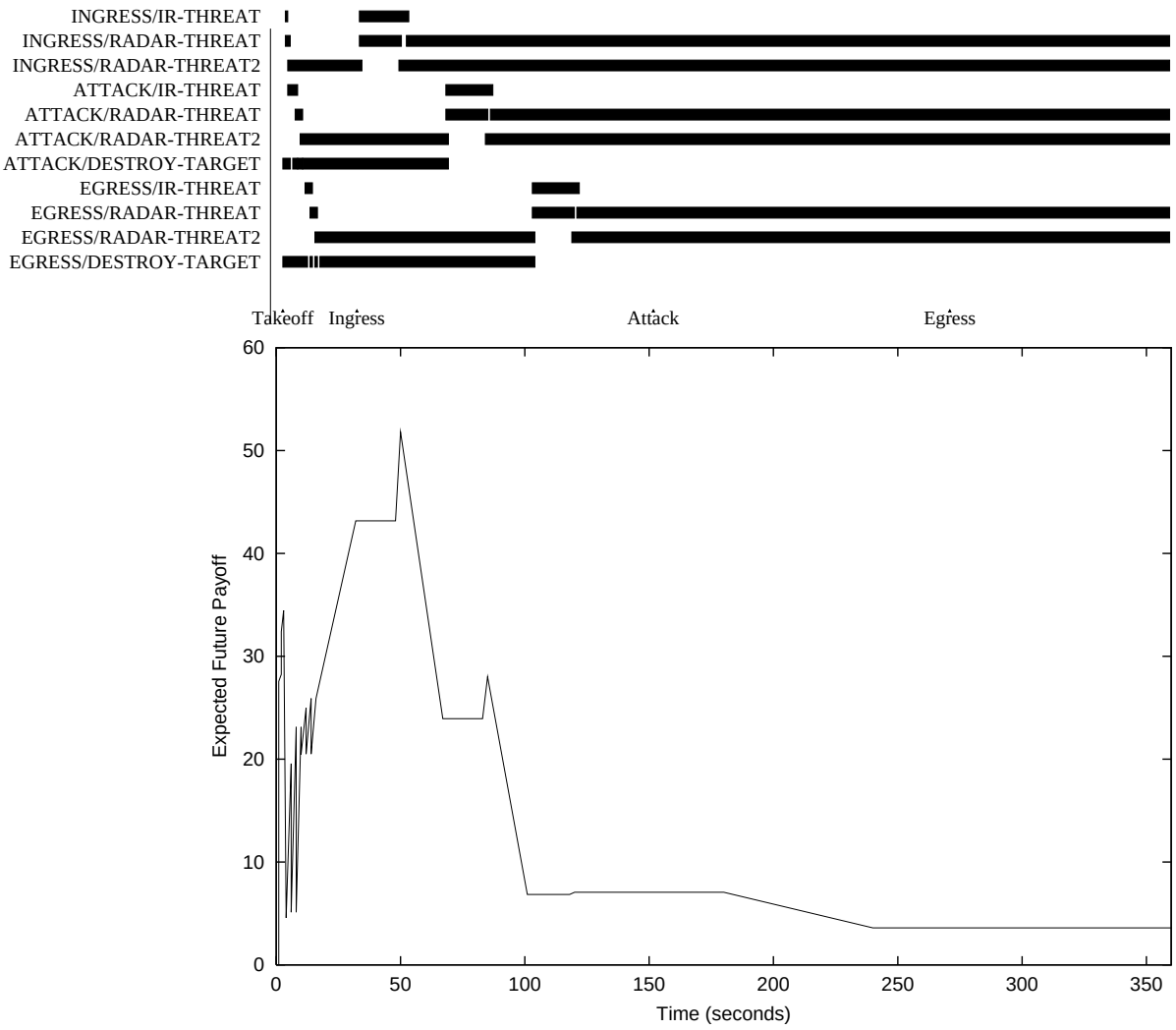


Figure 18. Gantt chart of threat and goal coverage for Agent S throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent does not have plans to destroy the targets during the appropriate phases, and thus acquires few utils.

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 18, this would look something like a white staircase descending to the right, with dark bars above.

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 18, 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 19 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. 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).

As shown in Figure 19, 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

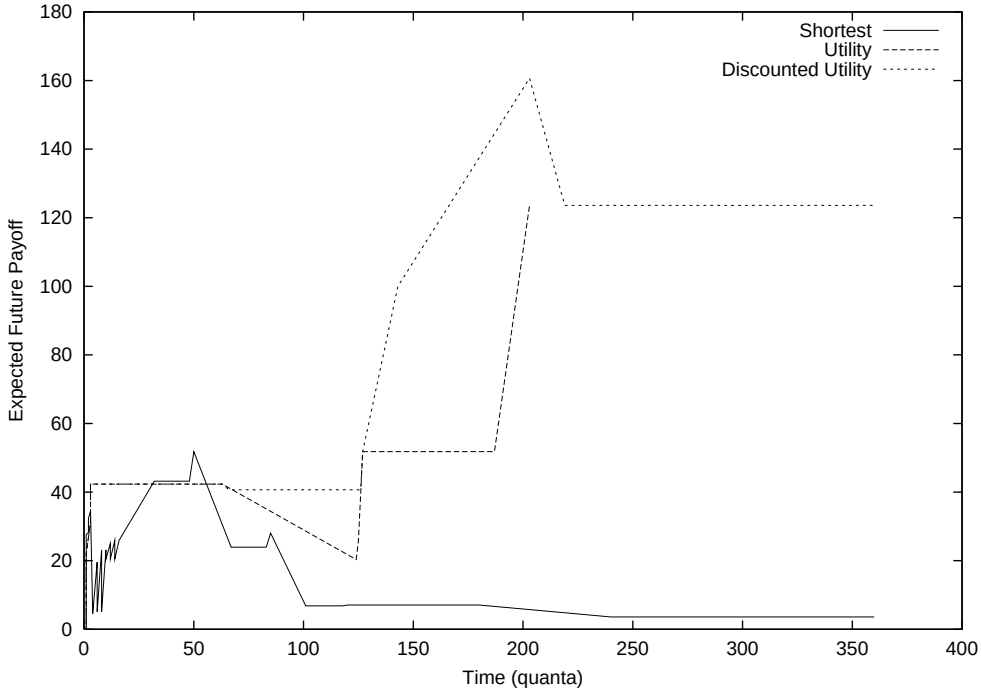


Figure 19. Each agent's expected payoff over the course of the mission.

cost-estimation function.

13.2.3. 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 20. 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,

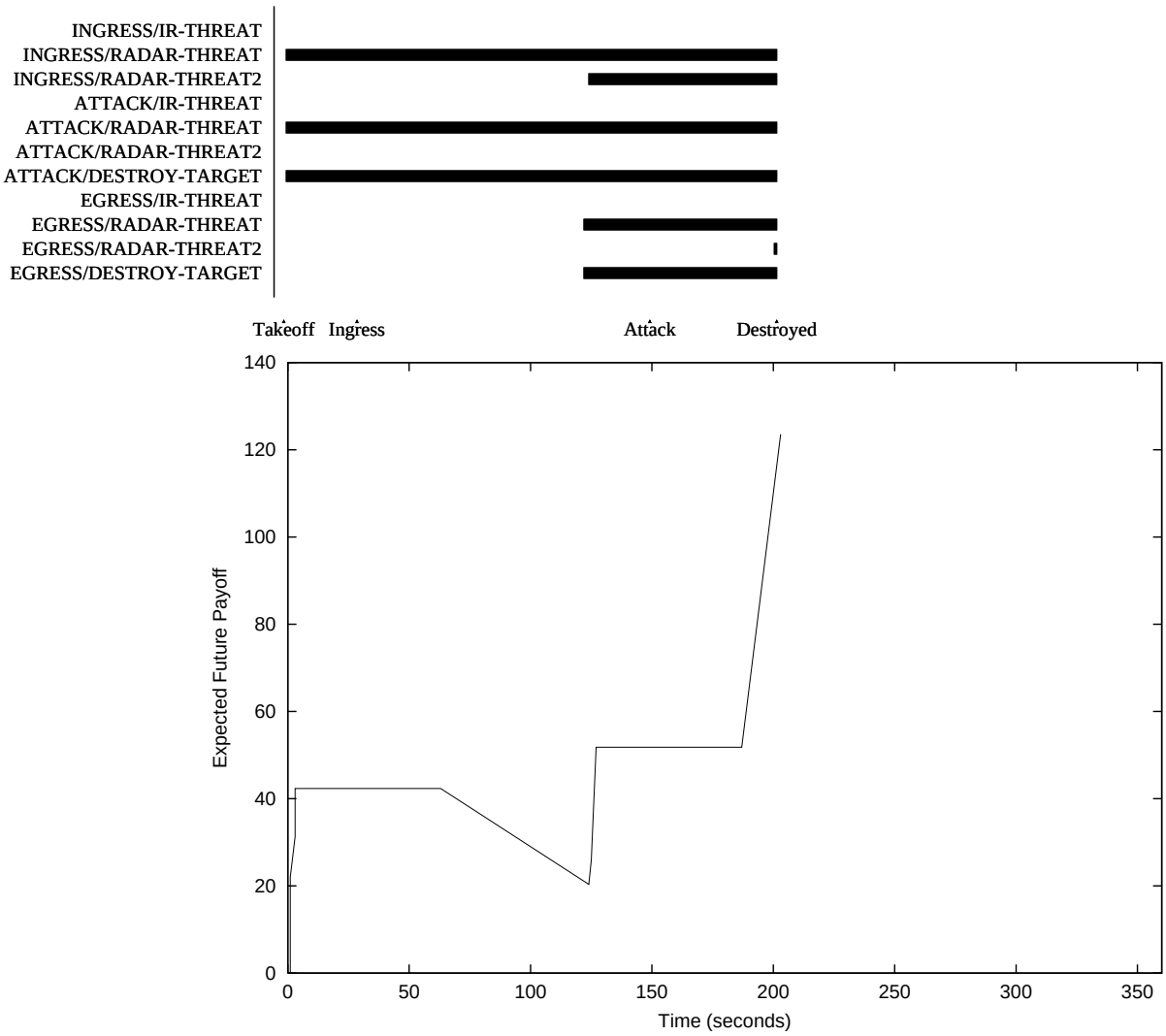


Figure 20. Gantt chart of threat and goal coverage for Agent U throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent is not prepared to defeat `radar-threat2` in the attack phase, and it is destroyed.

it fails to develop a good defensive plan, and is destroyed by a radar-guided missile[5].

13.2.4. Analysis of Agent Agent DU

Figure 21 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.

13.3. Demo 3: Hard-Real-Time Coordination

In addition to coarse-grain multi-agent negotiation over roles and responsibilities as illustrated in Section 13.1, MASA-CIRCA is also capable of a much finer-grain, hard-real-time coordination that we call "coordinated preemption." By *coordinated preemption*, we mean a set of plans that can be executed by distributed agents to detect, react to, and defeat a threat. For example, suppose one UAV has a sensor that can detect a missile that has been launched at the team, but it has no countermeasures to defeat a missile (perhaps because of weight/power constraints or earlier damage). Furthermore, suppose that another UAV has the appropriate ECM to defeat the threatening missile, but it has no threat-detection sensors. How can the two distributed agents build their plans to accomplish a coordinated preemption: "**You sense** the threat and **I'll act** to defeat it"?

As described in [16], we have developed fully automatic techniques for planning and executing this type of coordinated plan. The demonstration video available at ../movies/2plane-senseact-v2.avi[6] illustrates this type of coordination. The demonstration video involves two aircraft with heterogeneous capabilities:

Lead — The lead aircraft (colored blue) can sense targets but has no weapons to shoot them. It can also defend the team against IR-guided missiles (by popping flares and leading evasive maneuvers) but it cannot directly sense when an IR-guided missile is threatening the team.

Wing1 — The single wingman aircraft has a sensor that can detect IR-guided missiles, but it cannot defend against them. It also has air-to-surface missiles that it can fire at ground targets, but it cannot directly detect the targets.

Naturally, the demonstration involves these aircraft encountering both IR-guided missile threats and targets, and performing the appropriate communication actions to notify each other of the world features they cannot sense directly.

[5] 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.

[6] Note that this .avi requires onetime installation of a codec, available at ../movies/tscc_codec.exe.

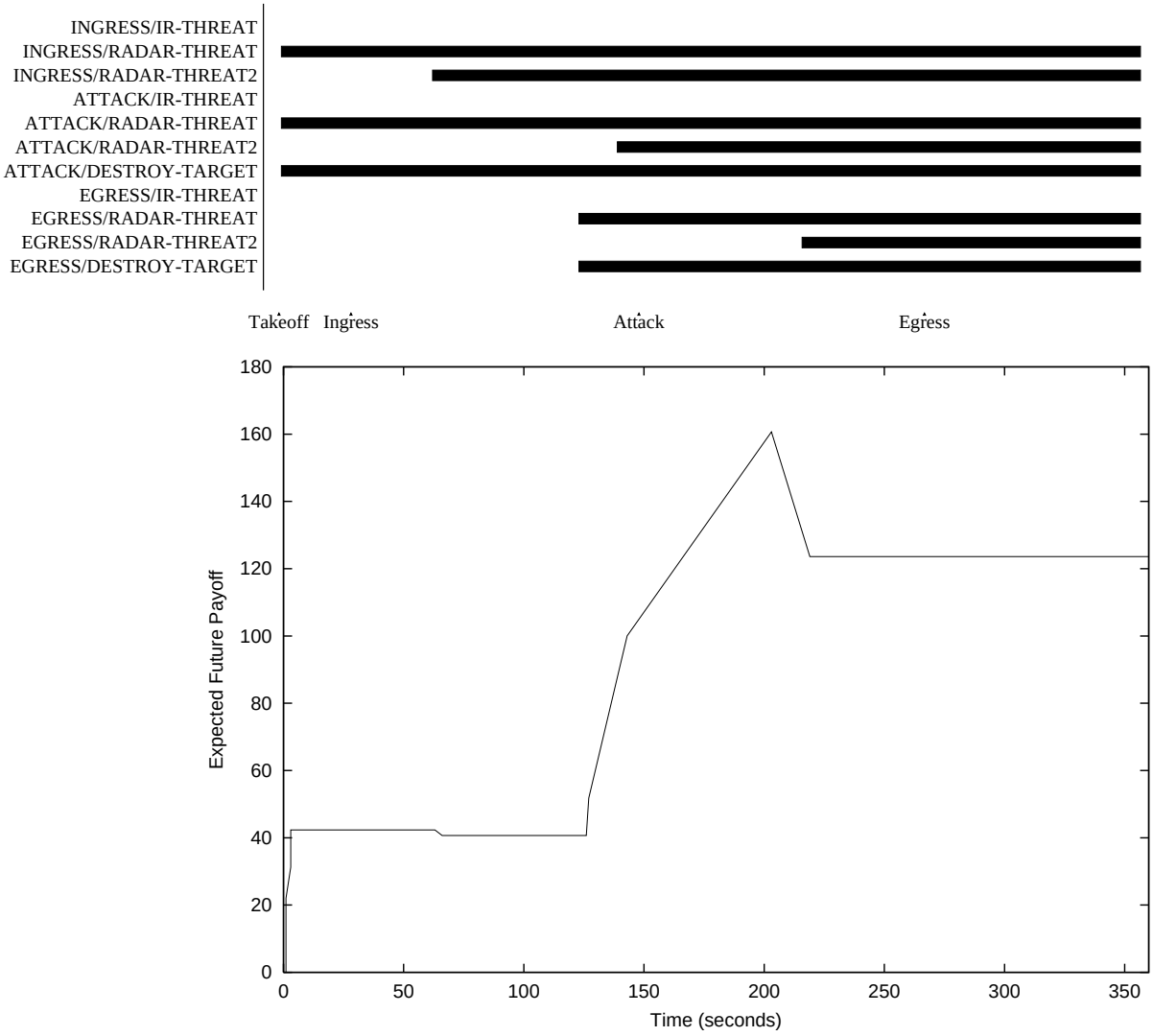


Figure 21. Gantt chart of threat and goal coverage for Agent *DU* throughout the mission, along with graph of expected future utility corresponding to plan coverage. Note that the agent has plans to accomplish the `destroy-target` goals by the time the respective phases occur.

13.4. Demo 4: Execution Monitoring and Responding to Unexpected Threats/Goals

This demonstration illustrates how CIRCA can both detect and respond to unexpected situations (that is, situations that are not handled by its current best plans). There are several ways in which an unexpected situation ("state") can arise:

AMP Pruning (Deliberation Scheduling) — As the AMP determines what mission phases and associated problem configurations should be planned for, it includes and omits various threats and goals. For example, suppose that the system is planning for a UAV mission phase in which two types of threats and two goals are potentially relevant. The AMP may decide to quickly build a one-threat, zero-goal plan. The resulting executable plan would not be able to achieve any goals or defeat one of the threats. However, if the unhandled threat actually arose, the system could still recognize this situation and modify its understanding of the likelihood of the threat (now definite), and potentially build a new plan to rapidly respond to the situation.

CSM Pruning (Planning) — All versions of the CSM can prune out regions of the state space from consideration, either by making states unreachable via preemption or by simply ignoring states as too-improbable. When the CSM has finished building a plan by assigning actions to all reachable states, it also has an implicit (non-enumerated) description of all of the unreachable states.

Model Gaps — The most fundamental types of unexpected states may arise if the models of state (features and their possible values) are incomplete. CIRCA cannot detect or respond to situations that are completely outside of its domain-supplied model representation or primitive sensing capabilities. In other words, CIRCA can only recognize and distinguish situations by the sensing primitives it is given in a domain; it cannot create new sensors. If two states differ only in unobservable characteristics, CIRCA will treat them the same. If the actions CIRCA takes result in unmodeled nondeterministic behavior because of the unobservables, the system will encounter a "model mismatch." This is a quite common restriction of autonomous control systems: we assume that they are not going to have to hypothesize undetected influences and then construct, presumably from external artifacts, sensors that make those influences detectable. Interestingly, it should be possible to extend CIRCA to perform exactly that type of functionality, if desired.

One reason the RTS is highly reactive and does not try to track the CSM's world model is that this allows it to continue operating past model mismatches. Suppose, for example, that the CSM's world model says that taking action A in state X will lead to state Y. But, due to unobservable, unmodeled influences, when the RTS actually takes the action A in state X it sometimes leads to a different state Z. No problem; since the RTS is not tracking the CSM world model and it is not checking to make sure that A resulted in Y. Rather, it simply takes the world as its own best

Phase:	Ingress	Attack	Egress
Threats	(radar-threat2 0.5) (radar-threat 0.5) (ir-threat 0.4)	(radar-threat2 0.5) (radar-threat 0.5) (ir-threat 0.4)	(radar-threat2 0.5) (radar-threat 0.5) (ir-threat 0.4)
Goals	(destroy-target 0.01)	(destroy-target 0.8)	(destroy-target 0.5)

Figure 22. The threat/goal mission profile. The UAV encounters the low-probability target and an unhandled threat in the Ingress phase, leading to two on-the-fly replanning episodes.

model [2] and moves on to react to Z. While this is not guaranteed to be the best strategy, it is the best the system can do with its limited knowledge.

If model mismatches were actually detected, they could be an opportunity for learning and model revision; this remains an area for future work.

13.4.1. Demo Scenario

This scenario is a modified version of the deliberation scheduling scenario. A single aircraft flies a mission that is described as having a series of potential threats and goals, with associated probabilities. However, some of the goals and threats that are actually encountered during the mission were not considered likely to occur, and thus CIRCA is unlikely to have planned for them when they arise. Figure 22 shows an outline of the expected threats/goals by phase. Threats are listed with their "lethality" rating, which in this case is the probability that, if CIRCA does not plan for them, they will kill the agent during this mission phase. This lethality value subsumes several constituent probabilities that could be defined in a more refined model (e.g., the probabilities that the threat will be encountered, that it will fire one or more times, and that the weapon(s) will kill the UAV). Goals are listed with their "opportunity probability." This is the probability that the agent will encounter a situation in which it is possible to achieve the goal (e.g., that a killable target will be detected).

For this demonstration, we see that the threat profile is the same across each phase: `radar` and `radar2` threats are equally likely, and `ir-threats` are somewhat less likely. However, the probability of encountering a target that can be destroyed (the opportunity probability of the `destroy-target` goal) is very low in the ingress phase, high in the attack phase, and moderate in the egress phase.

As with the other demonstrations, the aircraft begins flying the mission as soon as baseline plans are in place for each of the phases; the baseline plans do not address any of the threats or main mission goals, but simply accomplish the basic fly-to-waypoint behavior and transition to the next phase plan when the appropriate waypoint is reached. As a result, all of the planning for handling threats/goals is handled under time pressure, while the aircraft flies through the mission waypoints. The AMP's approximate

decision-theoretic deliberation scheduling algorithm decides what planning problems are addressed at any time.

13.4.2. Demo Script

In this section, we provide a time-annotated description of the scenario execution captured in the video file ../movies/solo-1.avi[7]. The time annotations, shown in minutes and seconds, are drawn from the video, and are thus only approximations.

Time 00:01 : Baseline plans complete; planning for Ingress/radar-threat — After the baseline plans are finished, and the AMP has downloaded them to the RTS, it sends a special "kickstart" schedule to the RTS. The kickstart schedule awakens the RTS and transfers control to the first baseline plan, for the Takeoff phase. This plan simply starts the plane's engines and, when sufficient speed is achieved, launches the plane into the air. Meanwhile, the AMP has decided to quickly build a plan to handle the potential radar-threat in the Ingress phase, which is rapidly approaching. The AMP was originally told the Takeoff phase might last 10 seconds; it actually takes 3 seconds for the plane to start moving, and after that it is considered in the Ingress phase.

Time 00:02 : Ingress/radar-threat plan complete; planning for Attack — Once the single-threat plan is complete, the deliberation scheduling algorithm decides to build a plan for the attack phase that will actually kill the target if it appears, so that the mission will achieve the potential reward of 200 utils. Handling the potential Attack/radar-threat is fairly easy at the same time as the target, so it chooses this combination for this planning problem.

Time 00:03 : Attack plan complete; planning for Ingress/radar and radar2 — With a reward-achieving plan in place for Attack, the deliberation scheduling algorithm next tries to improve the probability of surviving to get to the Attack phase by trying to plan for the two more likely threats in the current (Ingress) phase. The domain-specific CSM performance profile predicts this will take up to 20 seconds, and there are still over 100 seconds left in the expected Ingress phase duration. Note that the single-threat plan includes iftime TAPs monitoring for the presence of the unhandled threats and goals. The iftime server TAP is also in the guaranteed TAP schedule, so that the iftime TAPs will be executed in round-robin fashion, with at least one iftime TAP executing per schedule cycle.

Time 00:04 : View shows aircraft — The flight simulation visualization does not show the aircraft until the user selects a viewing position, so although the mission began four seconds ago, the plane is not visible in the movie until after the demonstration operator had a chance to mouse over the simulation window and hit 'f' to select the "follow behind" viewpoint.

Time 00:18 : Ingress two-threat plan complete; planning for two Attack threats — The CSM has finished planning earlier than expected, and the deliberation

[7] Note that this .avi requires onetime installation of a codec, available at ../movies/tscc_codec.exe.

scheduling algorithm re-focuses its attention on the Attack phase, seeking a safer plan that also addresses the destroy-target goal. The performance profile estimates that this planning task may take 60 seconds.

Time 00:30 : Unhandled IR-threat occurs — Before the two-threat Attack plan can be completed by the CSM, the TAP monitoring for unhandled threats detects an IR missile launch. The RTS sends a state update message to the AMP, which immediately halts the CSM, discards its outdated plans for the current phase, adjusts the IR missile threat lethality to 1.0 in the Ingress phase[8], and re-invokes its deliberation scheduling algorithm. The deliberation scheduling algorithm recognizes that it will not survive this phase without a plan for the IR threat, and tries to build the cheapest single-threat plan. A key point here is that the deliberation scheduling makes the correct decision in this situation without any hints, domain-specific or situation-specific heuristics, or other unprincipled methods. Knowing that a high-lethality threat is occurring leads the system to select the shortest-first approach. In other situations, the algorithm may select the longest task first, or other more complex choices.

Time 00:31 : IR-threat plan completes; planning for IR and radar threats — In less than a second, the CSM builds the single-threat plan for IR threats, downloads it to the RTS, and the RTS begins executing it. The revised plan causes the RTS to begin dropping flares to decoy the IR missile. Meanwhile, the AMP decides to plan to handle both IR threats (which it knows are occurring) and radar threats, which are also quite hazardous in this phase.

Time 00:35 : IR-threat defeated — The IR missile explodes on the last flare deployed by the aircraft.

Time 00:37 : Unhandled radar-threat occurs — A radar missile threat launches on the aircraft, which has not yet finished the two-threat plan. In less time than it takes the video's AID display to update, the AMP plans to handle that threat alone, downloads and begins executing the new plan (leading to evasive maneuvers) and begins planning to handle the existing radar threat along with potential radar2 type threats.

Time 00:41 : Radar-threat defeated — The radar missile explodes, just missing (behind) the evading aircraft. The aircraft stops evasive maneuvers and aims for its next waypoint.

Time 00:50 : Target appears (unhandled goal) — The current Ingress plan does not handle the destroy-target goal because the opportunity probability is so low; i.e., the input mission description did not think a target would be encountered during Ingress. However, the current plan does include an iftime TAP monitoring for the opportunity conditions (in this case, that the UAV has a missile and that a target is

[8] Here we can see it would be better to have separate probabilities for threat occurrence, which would be updated to 1.0, and threat lethality-if-encountered, which should remain unchanged. With those two composed together in the current version, we over-estimate the threat lethality-if-encountered. However, for our simplified simulation this is a sufficiently accurate model.

detected). When the target appears, the RTS again sends a state update to the AMP, in case an updated plan is required. Note that the RTS only sends state updates when an unexpected state occurs or when the AMP requests one. Observing that the state presents an opportunity to achieve the destroy-target goal, the AMP discards its current phase plans and modifies the goal's opportunity probability to 1.0. Again reinvoking its deliberation scheduling algorithm, the AMP decides it should build a plan to handle the goal, since acquiring its reward (200 utils) early in the plan will ensure at least some overall mission utility. The plan to handle the target and a potential radar-threat is created in under a second and the AID display shows the target will be handled.

Time 00:51 : UAV launches at target; planning for Attack phase— With a new destroy-target plan downloaded to the RTS, the UAV fires at the target while the AMP/CSM work to build a more complex Attack phase plan to handle two threats and the destroy-target goal. The plan to destroy a target is quite simple; the RTS merely needs to invoke a single fire-missile primitive. This is a significant simplification of a true combat mission, which might involve rerouting and various complex targeting actions and weapon-release maneuvers. However, the demonstration is meant to illustrate the detection of unexpected threats/goals and the resulting intelligent selection of replanning objectives. The complexity of the (re)planning can vary, and could include other mechanisms such as route planners, simply by modifying the AMP's planner performance profile models.

Time 01:00 : Handled radar-threat occurs — The aircraft encounters a radar threat that is handled by the current plan, and it evades in the normal fashion without interrupting the AMP's planning process.

Mission continues per usual — After the target is destroyed, the mission proceeds as in the prior demonstrations, so we will omit further description. The video is truncated for compactness.

13.4.3. CIRCA Features Highlighted

This demonstration highlights several new features of CIRCA, including:

RTS Feedback — The RTS is now automatically synthesized with a built-in primitive `send-state-to-amp` that captures the current value of all system sensors and sends them back to the AMP as a snapshot of the world state. The AMP can then use this state description as the start state for a revised plan.

Unexpected threat detection — For each new plan, the AMP automatically synthesizes an if-time TAP that detects the preconditions of any threat (TTF) that is not handled by the plan.

Unexpected goal detection — For each new plan, the AMP automatically synthesizes an if-time TAP that detects the opportunity-conditions of any goal that is not handled by the plan. In the current simplified interpretation, the opportunity-conditions are defined as the preconditions of a single action that

achieves the goal. A more complete approach would involve recognizing that chains of reliable temporal transitions and actions may lead from an opportunity to the goal, and hence that states more than a single action away from the goal should be interpreted as goal opportunities.

Interruptible CSM— When running the CSM in the same Lisp environment as the AMP, the CSM is executed in a separate Lisp multiprocessing thread. This allows the AMP to remain attentive to messages from the RTS, including the state updates sent by the RTS's `send-state-to-amp` primitive. When a state update arrives, the AMP can interrupt the current CSM planning thread and initiate a different planning process, to quickly respond to threats/goals.

Deliberation Scheduling— As described in Section 11, the AMP's deliberation scheduling algorithm uses a myopic approximation to decision theoretic solutions, so that it does not require brittle situation-specific rules or heuristics. As described above, the deliberation scheduling algorithm smoothly balances consideration of time pressure (tending towards shortest-first planning), self-preservation (tending towards threat-first planning), and goal achievement (tending towards goal-first planning).

14. Conclusions

The MASA-CIRCA project has made major advances in several important technology areas, including:

- Real-time deliberation scheduling and automatic resource-customized problem formulation.
- Recognizing and responding to unexpected situations.
- Tightly-coupled multi-agent coordination.
- Incremental verification of plans as timed automata.
- Planning in probabilistic adversarial domains.
- Automatic problem space partitioning.
- Coordination protocols to reduce uncertainty about joint behaviors.
- Solving resource-constrained MDPs.

These planning techniques will form key elements of future autonomous, self-evaluating, self-adaptive control systems that operate robustly in mission-critical environments.

The references below include both papers generated by this project and by other projects and other authors. All of the related publications from Honeywell are available online at `http://www.cs.umd.edu/users/musliner`, while the University of Michigan publications are available at `http://ai.eecs.umich.edu/people/durfee/durfee.html`. All of the contract-funded publications are also contained on the CD-ROM within which this report was delivered to the government.

References

[1] R. Alur, "Timed Automata," in *Working Notes of the NATO-ASI Summer School on Verification of Digital and Hybrid Systems*, 1998.

[2] R. Brooks, "Intelligence without Representation," *Artificial Intelligence*, vol. 47, pp. 139–159, January 1991.

[3] D. Cofer, E. Engstrom, R. Goldman, D. Musliner, and S. Vestal, "Applications of Model Checking at Honeywell Laboratories," *Lecture Notes in Computer Science*, vol. 2057, pp. 296–303, 2001. Available online as `http://www.cs.umd.edu/users/musliner/papers/mc-app.ps.gz`. Available on this disk as `../papers/mc-app.ps.gz`.

[4] D. L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," in *Automatic Verification Methods for Finite State Systems*, J. Sifakis, editor, pp. 197–212, Springer Verlag, Berlin, June 1989.

[5] D. A. Dolgov and E. H. Durfee, "Satisficing Strategies for Resource-Limited Policy Search in Dynamic Environments," in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, 3, pp. 1325–1332, 2002. Available online as `ftp://www.eecs.umich.edu/people/durfee/dolgov_satisficing02.pdf`. Available on this disk as `../papers/dolgov_satisficing02.pdf`.

[6] D. A. Dolgov and E. H. Durfee, "Approximating Optimal Policies for Agents with Limited Execution Resources," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003. Available online as `ftp://www.eecs.umich.edu/people/durfee/dolgov_approximating03.pdf`. Available on this disk as `../papers/dolgov_approximating03.pdf`.

[7] D. A. Dolgov and E. H. Durfee, "Constructing Optimal Policies for Agents with Constrained Architectures (abstract)," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 974–975, 2003. Available online as `ftp://www.eecs.umich.edu/people/durfee/dolgov_cmdp_poster03.pdf`. Available on this disk as `../papers/dolgov_cmdp_poster03.pdf`.

[8] D. A. Dolgov and E. H. Durfee, "Constructing Optimal Policies for Agents with Constrained Architecture," Technical Report CSE-TR-476-03, Electrical Engineering and Computer Science, University of Michigan, 2003. Available online as

ftp://www.eecs.umich.edu/people/durfee/dolgov_CSE-TR-476-03.pdf. Available on this disk as ../papers/dolgov_CSE-TR-476-03.pdf.

[9] E. H. Durfee, "Strategies for Discovering Coordination Needs in Multi-Agent Systems," *DoD Software Tech News*, vol. 5, no. 1, pp. 3–8, January 2002. Available online as ftp://www.eecs.umich.edu/people/durfee/Durfee02strategies.pdf. Available on this disk as ../papers/Durfee02strategies.pdf.

[10] M. Fröhlich and M. Werner, "Demonstration of the interactive Graph Visualization System daVinci," in *Proceedings of DIMACS Workshop on Graph Drawing '94*, R. Tamassia and I. Tollis, editors. Springer Verlag, 1995.

[11] 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. Available online as http://www.cs.umd.edu/users/musliner/papers/safer01.ps.Z. Available on this disk as ../papers/safer01.ps.Z.

[12] R. P. Goldman, D. J. Musliner, K. D. Krebsbach, and M. S. Boddy, "Dynamic Abstraction Planning," in *Proc. National Conf. on Artificial Intelligence*, pp. 680–686, 1997. Available online as http://www.cs.umd.edu/users/musliner/papers/aaai97.ps.Z.

[13] R. P. Goldman, D. J. Musliner, and M. J. Pelican, "Using Model Checking to Plan Hard Real-Time Controllers," in *Proc. AIPS Workshop on Model-Theoretic Approaches to Planning*, 2000. Available online as http://www.cs.umd.edu/users/musliner/papers/aipsws00.ps.Z. Available on this disk as ../papers/aipsws00.ps.Z.

[14] R. P. Goldman, M. J. Pelican, and D. J. Musliner, "Modeling and Verification for Automatic Synthesis of Real-time Controllers," in *Working Notes of the AAAI Spring Symposium on Real-Time Autonomous Systems*, 2000. Available online as http://www.cs.umd.edu/users/musliner/papers/ss00.ps.Z. Available on this disk as ../papers/ss00.ps.Z.

[15] F. Kabanza, M. Barbeau, and R. St.-Denis, "Planning Control Rules for Reactive Agents," *Artificial Intelligence*, vol. 95, no. 1, pp. 67–113, August 1997.

[16] K. D. Krebsbach and D. J. Musliner, "You Sense, I'll Act: Coordinated Preemption in Multi-Agent CIRCA," in *Working Notes of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, 2001. Available online as http://www.cs.umd.edu/users/musliner/papers/fs01a.ps.Z. Available on this disk as ../papers/fs01a.ps.Z.

[17] H. Li, "Execution Resource Allocation for a Real-Time Controller," in *Execution Resource Allocation for Distributed Real-Time Controllers*, chapter 4, Ph.D. Thesis, The University of Michigan, Ann Arbor, In preparation. Available online as ftp://www.eecs.umich.edu/people/durfee/li03thesis.pdf. Available on this disk as ../papers/li03thesis.pdf.

[18] H. Li, E. M. Atkins, E. H. Durfee, and K. G. Shin, "Practical State Probability Approximation for a Resource-Limited Real-Time Agent," in *Working Notes of the IJCAI-01 Workshop on Planning with Resources*, pp. 34–42, 2001. Available online as `ftp://www.eecs.umich.edu/people/durfee/li01probability.pdf`. Available on this disk as `../papers/li01probability.pdf`.

[19] H. Li, E. M. Atkins, E. H. Durfee, and K. G. Shin, "Resource Allocation for a Limited Real-Time Agent (abstract)," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1050–1051, 2003. Available online as `ftp://www.eecs.umich.edu/people/durfee/li03probability.pdf`. Available on this disk as `../papers/li03probability.pdf`.

[20] H. Li, E. H. Durfee, , and K. G. Shin, "Multiagent planning with internal resource constraints," in *Proceedings of the AAAI 2002 Workshop on Planning With and For MultiAgent Systems*, pp. 31–38, 2002. Available online as `ftp://www.eecs.umich.edu/people/durfee/li02convergence.pdf`. Available on this disk as `../papers/li02convergence.pdf`.

[21] H. Li, E. H. Durfee, and K. G. Shin, "Multiagent Planning for Agents with Internal Execution Resource Constraints," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 560–567, 2003. Available online as `ftp://www.eecs.umich.edu/people/durfee/li03convergence.pdf`. Available on this disk as `../papers/li03convergence.pdf`.

[22] D. McDermott, "Using Regression-match graphs to control search in planning," *Artificial Intelligence*, vol. 109, no. 1 – 2, pp. 111–159, April 1999.

[23] D. J. Musliner, "Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis," in *Proc. Int'l Workshop on Self-Adaptive Software*, 2000. Available online as `http://www.cs.umd.edu/users/musliner/papers/safer00.ps.Z`. Available on this disk as `../papers/safer00.ps.Z`.

[24] D. J. Musliner, "Planner Feedback: NIL is Not Enough," in *Working Notes of the AAAI Workshop on Representational Issues for Real-World Planning Systems*, 2000. Available online as `http://www.cs.umd.edu/users/musliner/papers/aaaiws00.ps.Z`. Available on this disk as `../papers/aaaiws00.ps.Z`.

[25] D. J. Musliner, "Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis," in *Lecture Notes in Computer Science*, number 1936, Springer-Verlag, 2001.

[26] D. J. Musliner, "Safe Learning in Mission-Critical Domains: Time is of the Essence (Extended Abstract)," in *Working Notes of the AAAI Spring Symp. on Safe Learning Agents*, March 2002. Available online as

http://www.cs.umd.edu/users/musliner/papers/ss02.ps.Z. Available on this disk as ../papers/ss02.ps.Z.

[27] 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. Available online as http://www.cs.umd.edu/users/musliner/papers/musliner-aij.ps.Z.

[28] D. J. Musliner, R. P. Goldman, and K. D. Krebsbach, "Deliberation Scheduling Strategies for Adaptive Mission Planning in Real-Time Environments," in *Proc. Third International Workshop on Self Adaptive Software*, 2003. Available online as http://www.cs.umd.edu/users/musliner/papers/safer03.ps.gz. Available on this disk as ../papers/safer03.ps.gz.

[29] D. J. Musliner, R. P. Goldman, and M. J. Pelican, "Using Model Checking to Guarantee Safety in Automatically-Synthesized Real-Time Controllers," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2000. Available online as http://www.cs.umd.edu/users/musliner/papers/icra00.ps.Z. Available on this disk as ../papers/icra00.ps.Z.

[30] D. J. Musliner, R. P. Goldman, and M. J. Pelican, "Planning with Increasingly Complex Executive Models," in *Proc. Int'l Conf. on Intelligent Robots and Systems*, 2001. Available online as http://www.cs.umd.edu/users/musliner/papers/iros01.ps.Z. Available on this disk as ../papers/iros01.ps.Z.

[31] D. J. Musliner and K. D. Krebsbach, "Multi-Agent Mission Coordination via Negotiation," in *Working Notes of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, 2001. Available online as http://www.cs.umd.edu/users/musliner/papers/fs01.ps.Z. Available on this disk as ../papers/fs01.ps.Z.

[32] D. J. Musliner, M. J. S. Pelican, and K. D. Krebsbach, "Building Coordinated Real-Time Control Plans," in *Proc. Third Annual International NASA Workshop on Planning and Scheduling for Space*, October 2002. Available online as http://www.cs.umd.edu/users/musliner/papers/nasaws02-senseact.ps.gz. Available on this disk as ../papers/nasaws02-senseact.ps.gz.

[33] M. Pistore and P. Traverso, "Planning as Model Checking for Extended Goals in Non-deterministic Domains," in *Proc. Int'l Joint Conf. on Artificial Intelligence*, pp. 479–486, 2001. Available online as http://citeseer.nj.nec.com/pistore01planning.html.

[34] M. Sze and E. H. Durfee, "Resource-Bounded Subgoaling: Techniques for Automated Mission Phasing," Technical report, Electrical Engineering and Computer Science, University of Michigan, Unpublished. Available online as ftp://www.eecs.umich.edu/people/durfee/Sze01meganode.pdf. Available on this disk as ../papers/Sze01meganode.pdf.