Detecting and Reacting to Unplanned-for World States

Ella M Atkins
University of Michigan
263 PUBLICATIONS **5,882** CITATIONS

Edmund H. Durfee
University of Michigan
345 PUBLICATIONS **9,082** CITATIONS

Detecting and Reacting to Unplanned-for World States[1]

Ella M. Atkins **Edmund H. Durfee** **Kang G. Shin**

University of Michigan AI Lab
1101 Beal Ave.
Ann Arbor, MI 48109
{marbles, durfee, kgshin}@umich.edu

Abstract

The degree to which a planner succeeds and meets response deadlines depends on the correctness and completeness of its models which describe events and actions that change the world state. It is often unrealistic to expect perfect models, so a planner must be able to detect and respond to states it had not planned to handle. In this paper, we characterize different classes of these "unhandled" states and describe planning algorithms to build tests for, and later respond to them. We have implemented these unhandled state detection and response algorithms in the Cooperative Intelligent Real-time Control Architecture (CIRCA), which combines an AI planner with a separate real-time system so that plans are built, scheduled, and then executed with real-time guarantees. Test results from flight simulation show the new algorithm enables a fully-automated aircraft to react appropriately to certain classes of unhandled states, averting failure and giving the aircraft a new chance to achieve its goals. We analyze CIRCA's capability to accommodate challenging domain characteristics, and present flight simulation examples to illustrate how CIRCA handles each.

1 Introduction

Autonomous control systems for real-world applications require extensive domain knowledge and efficient information processing to build and execute situationally-relevant plans of action. To enable guarantees about safe system operation, domain knowledge must be complete and correct, plans must contain actions accounting for all possible world states, and response times to critical states must have real-time guarantees. Practically speaking, these conditions cannot be met in complex domains, where it is infeasible to preplan for all configurations of the world, if indeed they could even be enumerated. Realistic autonomous systems use heuristics to bound the expanded world state set, coupled with reactive mechanisms to compensate when unexpected situations occur.

In this paper, we focus on the question of how an autonomous system can know when it is no longer prepared for the world in which it finds itself, and how it can respond. We assume limited sensory and computational capabilities, and that a system will devote available resources to the accomplishment of its tasks. As a consequence, such a system will not notice unexpected occurrences in the world unless it explicitly has a task of looking for them. In other words, the system must satisfy absolute guarantees about safe operation in expected states, and must also be ready to recognize and respond to the unexpected.

To ground our discussion and empirically validate our solutions, we will consider issues in dealing with unexpected occurrences within the context of the Cooperative Intelligent Real-time Control Architecture (CIRCA) applied to the aircraft domain. CIRCA combines a traditional AI planner, scheduler, and real-time plan execution module to provide guaranteed performance for the control of complex real-world systems [Musliner]. With sufficient resources and accurate domain knowledge, CIRCA can build and schedule control plans that, when executed, are assured of responding quickly enough to any events so that CIRCA remains safe (its primary task) and whenever possible reaches its goals.

When faced with imprecise knowledge and limited resources, a CIRCA plan may not be prepared to handle all possible states. CIRCA may have planned actions to assure safety in some state but may not be able reach a goal state. Or, it may have anticipated a state but, due to resource limitations, chose not to schedule actions to keep it safe in that state. Or, it may not even have anticipated that state because of knowledge base imperfections. The contribution of this paper is to articulate, more precisely, such different classes of unhandled states (Section 3), to describe methods to detect when a system reaches one of these states (Section 4), and to respond appropriately (Section 5). We highlight how our algorithms improve CIRCA's performance for simulated aircraft control (Section 6), and discuss how CIRCA handles each of several challenging domain characteristics (Section 7).

2 Background

Ideally, a planner would examine all world states and build a universal plan [Schoppers] to handle all possible situations. However, this procedure may require exponential execution time [Ginsberg] and may not handle all situations when imperfect knowledge exists. Executive architectures such as CYPRESS [Wilkins] have been applied to problems similar to those discussed in this paper. CYPRESS concentrates on limiting planner execution time by restricting the planner's choices to high-level actions then requires runtime low-level action selection, so no response guarantees are possible. Conversely, CIRCA contains a deliberative planner that limits planning time by considering only those states

[1] This research was supported by NSF Grant IRI-9209031.

reachable from the initial state. CIRCA spends more time up-front building plans, assuming the set of reachable states is sufficiently small to reasonably bound execution time. However, CIRCA produces plans that are executed with real-time guarantees because all planned responses have been carefully scheduled in advance.

Figure 1 shows the general architecture of the CIRCA system. The AI subsystem (AIS) contains the planner and scheduler. The "shell" around AIS operations consists of meta-rules controlling knowledge areas, similar to the PRS architecture [Ingrand]. Working memory contains tasks to be executed, including planning, scheduling, downloading plans from AIS to real-time subsystem (RTS), and processing RTS feedback.

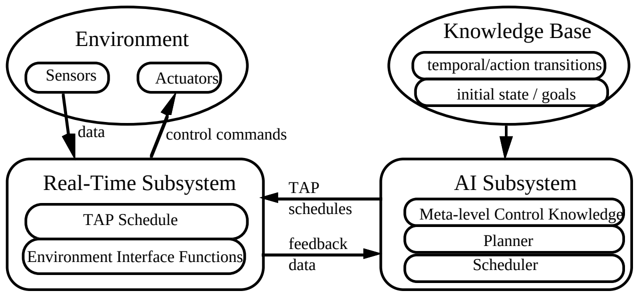

Figure 1. CIRCA Architecture.

The CIRCA domain knowledge base specifies subgoals which, when achieved in order, enable the system to reach its final goal. CIRCA plans separately for each goal in this list. During planning, the world model is created incrementally based on initial state(s) and available state transitions. The planner uses traditional methods of selecting actions based on estimated "cost vs. benefit" and backtracks if the action does not ultimately help achieve a goal or avoid failure. CIRCA minimizes memory and time usage by expanding only states produced by transitions from initial states or their descendants. State expansion terminates when the goal has been reached while avoiding failure states.

A CIRCA knowledge base contains two transition types: action and temporal. Action transitions correspond with commands explicitly executed by the RTS, while temporal transitions represent state changes not initiated by CIRCA. Each temporal transition has a probability function that models that transition's likelihood as a function of time, starting at the moment all preconditions become true. Time remaining until a transition will occur is particularly important when transition to failure is involved. In this case, CIRCA must schedule an action that is guaranteed to execute before the temporal occurs, preempting the transition to failure. CIRCA "plays it safe" by assuming the action must occur before the transition has more than small probability ε. The use of probabilistic models in CIRCA is described in [Atkins].

CIRCA's control plans are represented as cyclic schedules of test-action pairs (TAPs). Typical tests involve reading sensors and comparing sensed values with certain preset thresholds, while actions involve actuator commands or data transfer between CIRCA modules. When the planner creates a TAP, it stores an associated execution deadline, which is used by a deadline-driven scheduler [Liu] to create a periodic TAP schedule to guarantee system safety. If the scheduler cannot create a schedule to support all deadlines, the AIS backtracks to the planner. For the next iteration, the lowest probability temporal transitions are removed to reduce the number of actions planned. If the scheduler fails when only high-probability transitions are considered, the CIRCA planner fails, leaving the RTS executing its last plan which will ideally keep the system "safe" but never reach the goal.

Presuming the planner and scheduler are successful, the AIS downloads the TAP plan to the RTS. During normal operation, the RTS sends only handshaking messages to the AIS. This paper describes the introduction of RTS state feature feedback to prompt AIS replanning when an unhandled state is detected.

3 Unhandled State Classes

Figure 2 shows the relationship between subclasses of possible world states. Modeled states have distinguishing features and values represented in the planner's knowledge base. Because the planner cannot consider unmodeled states without a feature discovery algorithm, unmodeled states are beyond the scope of this paper. "Planned-for" states include those the planner has expanded. This set is divided into two parts: "handled" states which avoid failure and can reach the goal, and "deadend" states which avoid failure but cannot reach the goal with the current plan.

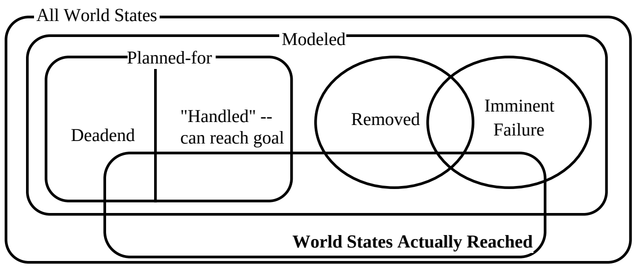

Figure 2. World State Classification Diagram.

A variety of other states are modelable by the planner. Such states include those identified as reachable, but "removed" because attending to them along with the "planned-for" states exceeds system capabilities. Other modeled states include those that indicate "imminent failure;" if the system enters these states, it is likely to fail shortly thereafter. Note that some states might be both "removed" and "imminent-failure", as illustrated in Figure 2. Finally, some modeled states might not fall into any of these categories, such as the states the planner considered unreachable but that are not necessarily dangerous. We are working to find other important classes or else show no other modelable state classes are critical to detect. As illustrated by the boldly outlined region in Figure 2, states actually reached may include any subclass. To assure safety, the set should only have elements in the "planned-for" region. When the set has elements outside this region, safety and performance depend on classifying the new state and responding appropriately. For this reason, we provide more detailed definitions of the most important classes.

A "deadend" state results when a transition path leads from an initial state to a state that cannot reach the goal, as shown in Figure 3. The deadend state is safe because there is no transition to failure. However, the planner has not selected an action that leads from this state via any path to the goal. As illustrated by a flight simulation example (Section 6), deadend states produced because no action can lead to a goal are called "by-necessity", while those produced because the planner simply did not choose an action leading to the goal are called "by-choice".

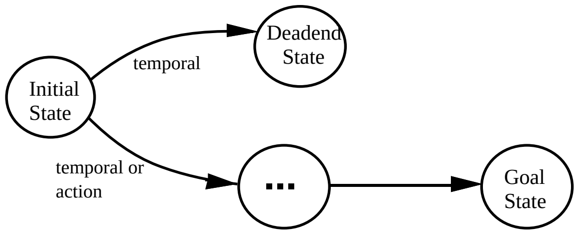

Figure 3. "Deadend state" illustration.

A planner that generates real-time control plans needs to backtrack whenever scheduling fails. When backtracking, the planner selects different actions while maintaining those required to avoid failure. However, even after exhaustive backtracking, a planner may fail to find actions that meet all objectives while still being schedulable. One option is ignoring some reachable states, thus not planning actions for them. A control plan so constructed cannot claim to be foolproof. However, for real-time control applications, it may be more important to make timing guarantees under assumptions that exceptional cases will not occur than to make no guarantees about a more inclusive set of cases.

One heuristic for selecting states to prune is to overlook the most unlikely states. A "removed" state set is created when the planner has purposefully removed the set of lowest probability states during backtracking, as illustrated in Figure 4. In the first planner iteration, all states with nonzero probability are considered, as depicted by the "Before Pruning" illustration. A low probability transition leads to a state which transitions to failure. This failure transition is preempted by a guaranteed action.

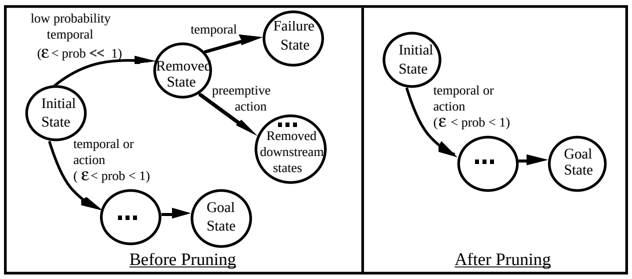

Figure 4. "Removed state" illustration.

Suppose the scheduler fails. The planner will backtrack and build a new plan without low-probability states. The resulting state diagram -- "After Pruning" -- is shown in Figure 4. Due to the low probability transition, all downstream states are removed from consideration. The preemptive action is no longer required, giving the scheduler a better chance of success. A flight simulation example with removed states is shown in Section 6.

During plan development, all temporal transitions to failure (TTF) from reachable states are preempted by guaranteed actions. If preemption is not possible, the planner fails. However, the planner does not worry about TTF from any states it considers unreachable from the initial state set. The set of all modelable states considered unreachable that also lead via one modeled temporal transition to failure are labeled "imminent-failure".[2] Actually reaching one of the recognizable imminent-failure states indicates either that the planner's knowledge base is incomplete or incorrect (i.e., it failed to model a possible sequence of states), or that the planner chose to ignore this state in order to make other guarantees.

Figure 5 shows a diagram of a reachable state set along with an isolated state (labeled "Imminent-failure") leading via one temporal transition to failure. This state has no incoming transitions from a reachable state, so the planner will not consider it during state expansion. However, if this state is reached, the system may soon fail. The imminent-failure unhandled states are important to detect because avoiding system failure is considered CIRCA's primary goal. Examples of imminent-failure states from flight simulation tests are described in Section 6.

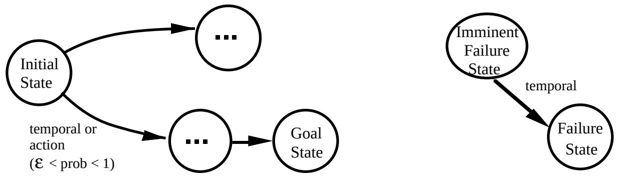

Figure 5. "Imminent-failure state" illustration.

4 Detecting Unhandled States in CIRCA

A critical premise in our work is that a planner cannot be expected to somehow just "know" when it has deviated from plans---it must explicitly plan actions and allocate resources to detect such deviations. For example, to make real-time guarantees, CIRCA's AIS must specify all TAPs to be executed, including any to detect and react to unhandled states. In our implementation, after the planner builds its normal plan, it builds TAPs to detect deadend, removed, and imminent-failure states. Other unhandled states, such as those "modeled" but outside "planned-for", "removed", and "imminent-failure" regions in Figure 2, are not detected by CIRCA. If reaching one of these unhandled states that is not detected by CIRCA, the system may eventually transition back to a planned-for state (where the original plan executes properly), transition to an imminent-failure state (where CIRCA will detect the state and react), or simply remain safe forever without reaching the goal. The algorithms to build lists for deadend, removed, and

[2] Note that it is also possible that states that are unmodelable could lead directly to failure with a known transition, or that states that are modelable could lead directly to failure with transitions that are not known to the planner, or that states that are not modelable could lead directly to failure with an unknown transition. We exclude these cases from the "imminent-failure" set because the planner is incapable of classifying them in this way.

imminent-failure states are described below. TAPs *could* include explicit tests for every set of features in that unhandled state list, but these tests would be repeated frequently during plan execution and may be time-consuming, as in PRS [Ingrand], where context checking could involve a large, non-minimal number of tests, including updates from sensors. In CIRCA, once each list is completed, the planner calls ID3 [Quinlan] with that unhandled state list as the set of positive examples and a subset of the reachable states (depending on unhandled state type) as the set of negative examples. ID3 returns what it considers a minimal test set, which is then used to detect that unhandled state class.

When any of the three unhandled state detection TAP tests are active, the RTS feeds back current state features to the AIS along with a message stating the type of unhandled state detected. The AIS then builds a new TAP plan which will handle this state. The algorithm by which the AIS replans is described in Section 5.

Deadend states cannot lead to a goal state. To identify this state set, CIRCA follows transition links from each state in search of a goal state. If no goal is found, that state is labeled "deadend" while the reachable states along a goal path are labeled "non-deadend". The deadend states are used as positive ID3 examples while non-deadend reachable states are used as negative examples.

Whenever backtracking due to scheduling difficulties or the planner's inability to find a valid plan, low-probability states are pruned. To build this "removed" state list, the planner executes its state expansion routine using the reachable state set as "initial states”. Planned action transitions are used to build new states. The result of this state expansion is a list of states containing both the original reachable states and the new low-probability or "removed" states that were not considered in the original plan. To build the removed state detection tests, ID3 is called with this “removed” state list (minus the original reachable states) used as positive examples and all original reachable states used as negative examples.

While the planner should look for deadend and removed states because they are more likely to occur than other unhandled states, likelihood is not the only criterion for allocating resources to detection. No matter how unlikely, detecting imminent-failure states is important because of the potentially catastrophic consequences of being in such states. When building imminent-failure state sets, we assume the modeled set of temporal transitions to failure (TTFs) is complete and correct, even though reaching such a state implies some other transition is not accurately modeled. The AIS begins with a list of all precondition feature sets from TTFs. This list is expanded to fully enumerate all possible states that would match these preconditions. Any reachable states are removed from this list. Minimized imminent-failure detection TAP tests are then built with this list as ID3 positive examples and the reachable states as negative examples. Note that a complete list of fully-instantiated states can be large with general preconditions leading to failure, so we hope to employ a Monte Carlo method to build an approximate set of ID3 examples, then use an anytime algorithm [Dean] with ID3 to truncate classification tree building when a planning deadline approaches. The generated TAP test will not be 100% accurate, but we believe estimates are necessary since the alternative is possible exponential execution time.

5 System Reaction to an Unhandled State

Regardless of whether it is a "deadend", "removed", or "imminent-failure" state, any unhandled state has the potential to prevent the RTS from ever reaching its goal. Additionally, removed or imminent-failure states will likely result in system failure if no action is taken. To increase CIRCA's goal achievement and failure avoidance, the AIS replans whenever unhandled states are detected.

When one of the special TAPs described in Section 4 detects any class of unhandled state, the RTS feeds back all state features to the AIS.[3] The AIS generates the equivalent of an interrupt that is quickly serviced in the following sequence. First, the AIS reads feedback message type (e.g., "deadend”). Next, the AIS reads the uplinked state features, selects a subgoal with preconditions matching the uplinked state features, and then runs the planner state expansion module, using state feature feedback as the initial state, all temporal transitions, and the executing plan's TAPs as action transitions. The returned state list contains all possible states the RTS could reach *while* the AIS is replanning,[4] thus each is a possible initial state when the new plan begins executing. The AIS then replans using this potentially large initial state set and the selected subgoal. This new plan is downloaded to the RTS which can then react to the previously unhandled state and its descendants (i.e., the constructed initial state set).

By detecting all unhandled states which may reach failure,[5] the system will always be able to initiate a reaction to avoid impending doom. However, this is predicated on the planner being able to return a control plan to avert disaster faster than disaster could strike. For the purposes of examples in this paper, we assume that this is the case: in the aircraft domain, we assume the plane is at sufficient altitude and distance from the runway that the new plan is constructed before the plane crashes. However, in other situations the system might have less opportunity to postpone disaster. We are currently working to build a more robust imminent-failure handling system by adding failure-avoidance actions to a control plan whenever possible and by limiting replanning time.

6 Flight Simulation Testing

We tested our unhandled state classification, detection, and reaction ideas in an aircraft flight simulator. Perhaps the main attraction to the aircraft domain is that continuous real-time operation is essential -- an aircraft can never "stop and remain safe indefinitely" once it has left the ground. The fully-automated flight problem has only been solved when an aircraft is restricted to certain regions of state-space. The "solved" part of flight can be modeled in a

[3] Even though the RTS senses feature values in sequence, we assume none will change before the state is uniquely determined.

[4] We assume no further modeling errors during this state expansion, thus no possible initial states are ignored.

[5] Presuming we have modeled all actual transitions to failure.

CIRCA domain knowledge base, then CIRCA's AIS can create plans that issue commands to a low-level control system. With the addition of unhandled state feedback, CIRCA begins to reach beyond traditional autopilots, allowing the system to select a new plan (e.g., trajectory) which may not reach the previous goal but that can avoid failure and divert to a new location if necessary. We interfaced the ACM F-16 flight simulator [Rainey] to a Proportional-Derivative (P-D) controller [Rowland] that calculates actuator commands to achieve a specific altitude and heading. Modeled state features include altitude, heading, location (or "FIX"), gear and traffic status, and navigation sensor data. CIRCA successfully controlled the aircraft during normal pattern flight (illustrated in Figure 6) from takeoff through landing.

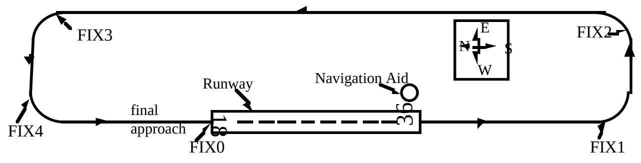

Figure 6. Flight pattern flown during testing.

We have tested our algorithms using two emergencies: "gear fails on final approach", and "collision-course traffic on final approach". In either situation, failure to notice and react to the problem will cause a crash. We included feature "gear" with values "up" and "down", and feature "traffic" with values "yes" and "no". Note that more complex features would work, but these examples illustrate the utility of our algorithms. We will discuss only the "gear failure" emergency here; similar results from "collision-course traffic" are discussed in [Atkins].

During normal flight, the gear was left down. A TTF occurred with gear "up" when landing to simulate the impending crash. Proper reaction to a "gear up" emergency is to execute a go-around (i.e., continue around the pattern a second time to avoid an immediate crash), performing any available actions such as cycling the gear lever to help the gear extend. A deadend "gear up" state was created using a temporal transition with precondition "gear down" and postcondition "gear up". Figure 7 illustrates the deadend state with action "hold positive altitude" to avoid failure. For brevity, two of seven state features are listed.

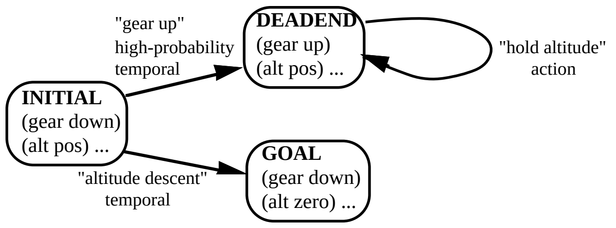
"gear up" high-probability temporal
DEADEND (gear up) (alt pos) ...
"hold altitude" action
INITIAL (gear down) (alt pos) ...
"altitude descent" temporal
GOAL (gear down) (alt zero) ...

Figure 7. Flight Simulation -- Deadend State.

A removed "gear up" state set was created with a low-probability "gear up" temporal transition and a time-consuming action to put the gear back down. The initial AIS plan included a "gear up" temporal along with a "gear down" action. This action was guaranteed for the final approach region, and the scheduler failed because the action when combined with other approach-to-landing operations was too time-consuming. The planner then pruned low-probability states, and they became "removed" states. Figure 8 shows a partial state diagram illustrating simplifications due to low-probability state removal.

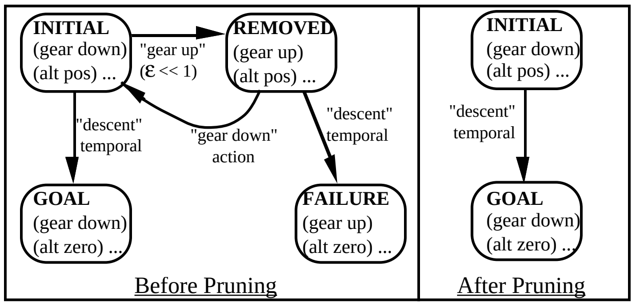
INITIAL (gear down) (alt pos) ...
"gear up" (ε << 1)
REMOVED (gear up) (alt pos) ...
"descent" temporal
"gear down" action
"descent" temporal
GOAL (gear down) (alt zero) ...
FAILURE (gear up) (alt zero) ...
Before Pruning
INITIAL (gear down) (alt pos) ...
"descent" temporal
GOAL (gear down) (alt zero) ...
After Pruning

Figure 8. Flight simulation -- Removed State.

An imminent-failure "gear up" state set was created by having an initial state with gear down and omitting temporal transitions from "gear down" to "gear up". Without a temporal transition to "gear up", the planner had no reason to expand states with a "gear up" feature, so no "gear up" action was planned. However, since attribute "gear up" led directly to failure, an imminent-failure-state detection TAP was built so the RTS could detect the "gear up" state if it occurred. Figure 9 shows the reachable state set and "gear up" imminent-failure state.

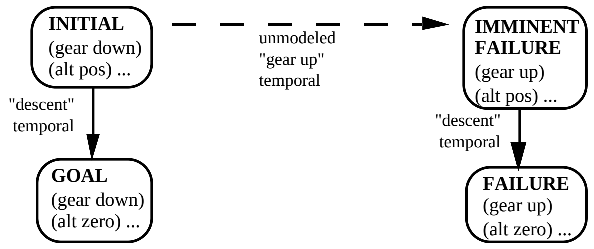
INITIAL (gear down) (alt pos) ...
unmodeled "gear up" temporal
IMMINENT FAILURE (gear up) (alt pos) ...
"descent" temporal
GOAL (gear down) (alt zero) ...
"descent" temporal
FAILURE (gear up) (alt zero) ...

Figure 9. Flight Simulation -- Imminent-failure States.

Test results for each unhandled state subclass are summarized in Table 1. For each test, the aircraft flew around the pattern, and the gear unexpectedly went up just after starting final approach past FIX4. The table describes events occurring after the gear goes up. Rows represent test runs, while columns describe test features. "Gear down action" and "Gear up temporal" columns depict whether those transitions were present in the knowledge base. Column 4 indicates whether unhandled state algorithms were used, followed by the "Detected state type" column indicating which unhandled state class (if any) was detected. Output columns show results and an indication of whether the plane crashed or landed safely. In summary, whenever "gear up" occurred but went undetected (tests 1 and 2), the aircraft crashed. If "gear up" was deadend "by necessity" (tests 3 and 4), the aircraft lands safely after a go-around only if gear serendipitously extends (which has happened on occasion). In all other cases (tests 5 - 7), detecting and reacting to the problem enables CIRCA to execute a go-around and subsequent "gear down" action, resulting in a successful landing.

Table 1. Simulation Test Results with "Gear Up" Emergency.

Input					**Output:**	
Test #	"Gear-down" action	"Gear up" temporal	Detection algorithm used	Detected state type	Result	Crash?
1	N/A	No	No	None	Lands "gear up" on runway	Yes
2	N/A	Yes	No	None	Flies straight ahead, avoiding "gear-up" landing until out of fuel	Yes
3	No	Yes	Yes	Deadend by necessity	Gear never extends; plane executes go-arounds until out of fuel	Yes
4	No	Yes	Yes	Deadend by necessity	Gear locks itself during go-around(s)	No
5	Yes	Yes	Yes	Deadend by choice	Plane executes "gear down" during go-around	No
6	Yes -- slow execution	Yes -- low probability	Yes	Removed	Plane executes "gear down" during go-around	No
7	Yes	No	Yes	Imminent-failure	Plane executes "gear down" during go-around	No

As shown in Table 1, detecting the "gear up" problem only increases the aircraft's chance to successfully land, particularly when the "gear down" action is functional, thus this simple example clearly illustrates how CIRCA performance can improve with unhandled state detection. However, "gear up" and "collision-course traffic" anomalies are relatively simple to model. We are working to show that CIRCA functions properly when dynamically complex or interrelated features produce unhandled states. This requires modification of the P-D controller because extreme flight attitudes are not controllable with such a simple linear system.

Despite numerous improvements to be made before our aircraft is "ready-to-fly" except in simulation, our tests demonstrate the advantages of our algorithms to detect and respond to unhandled states. We expect flight simulation to provide many challenges during future testing.

7 Domain Characteristics in CIRCA

The following paragraphs summarize how CIRCA performs with respect to each domain characteristic identified for this symposium. This paper has concentrated on improving CIRCA's performance in complex domains, so "complexity" is not discussed here.

Dynamism

As discussed in Section 2 and [Musliner], CIRCA domain knowledge bases contain two types of state transitions: action and temporal. Temporal transitions explicitly model world changes due to either other agents or exogenous events. These transitions have attributes that specify when they can occur (preconditions), how state features change (postconditions), and their probability as a function of time. Our flight simulation example contained many temporal transitions, such as the [improbable] transition from "gear down" to "gear up".

Concurrency

CIRCA handles action/event and event/event concurrency, but not action/action concurrency, since it requires some nonzero action execution time and uses a single processor for plan execution. As an example of how CIRCA handles concurrency, suppose an action is selected (or temporal transition occurs) from some state ("State1"), resulting in a new state ("State2"). However, suppose an immediate and inevitable (probability=1.0 at time=0) temporal transition leads from State2 to another state ("State3"). Since the system spends no time in State2, the planner considers no alternate paths from State2, and the resulting plan is identical to that with a "concurrent" transition from State1 directly to State3. We modeled no concurrent action/event (or event/event) during testing; however, we could easily model such pairs. For example, suppose CIRCA selected an action to extend the aircraft flaps when the aircraft velocity is above Vne ("never exceed speed"). There is nearly 100% probability that the event "flaps rip off" will immediately occur, thus the planner will effectively consider a path directly from "State1: flaps up, speed>Vne" to "State3: flaps ripped-off, speed>Vne", since "State2: flaps down, speed>Vne" will occur for zero time.

Uncertainty

As described in Section 2 and [Atkins], we model uncertainty via temporal transition probabilities. We do not explicitly model action uncertainty; we assume action postconditions become true if the action is initiated. However, if an action does not execute properly, the postconditions may be different than those specified by that transition. We can indirectly handle action execution uncertainty in a manner similar to that for concurrency. Rather than associate the uncertainty with the action taken to proceed from "State1" to "State2", we would model it as one (or more) immediate probabilistic temporal transition(s) from "State2". Such a transition could lead to a "State3" that reflects action misexecution results. For example, if the action simply did not execute, State3 would be the same as State1. This model for action uncertainty is not elegant, so we are interested in alternatives that will also preserve the time-dependent temporal transition uncertainties we handle now.

Goal Variability

Typically, goal states with no matching temporal transitions -- "achievement goals" because they remain true indefinitely -- require no actions, except terminating plan execution. We had such a goal during our simulation tests. The aircraft successfully reaches the "safe" state of sitting motionless on the ground after completing "flight around the pattern". Because the plane is "safe" and immobile, no actions are required so plan execution terminates.

The other extreme involves "maintenance" goals -- those that are reachable but require subsequent actions to maintain. In CIRCA, the planner identifies such goals by noticing outgoing temporal transitions. The planner then examines downstream states and selects a set of actions to eventually return the system to a goal state. To illustrate how CIRCA would handle this goal type, consider the aircraft task "execute a holding pattern". An ideal pattern is a point in space, but an aircraft must keep a forward velocity to stay aloft, so the goal is maintained by flying in a constant-altitude pattern, similar in shape to the pattern in Figure 6. CIRCA would select, say, a goal location of FIX4. After reaching FIX4, actions are selected to fly to FIX1, FIX2, FIX3, then FIX4 (the goal) cyclically until the aircraft is cleared to land.

Changing Objectives

For normal operation, CIRCA assumes the set of prespecified knowledge base goals is sufficient, so there are no algorithms for changing objectives. CIRCA's planner cannot dynamically change objectives during plan execution because it has little knowledge of the current world state, except when a goal state is reached or an unplanned-for state is fed back. We have incorporated one method for changing objectives when an unplanned-for state is detected. When encountering such a state, it is possible that the original goal is unreachable, so CIRCA's AIS may select a different goal that is reachable from the unplanned-for state. This situation is illustrated by the "gear up" tests in Section 6. The original "final approach" goal was to successfully land on the runway. However, when the gear retracted, this goal was no longer directly achievable because the airplane would crash, so the planner selected the new goal of flying to FIX1, then continuing around the pattern a second time.

We would like to incorporate more elegant methods for changing objectives, such as dynamically defining goals that are not in the knowledge base, particularly when the planner finds it difficult to reach any specified goal.

Action Interruptibility

In CIRCA, an action is executed whenever the current state matches the TAP test for that action. We currently execute actions by calling non-interruptable functions on CIRCA's RTS. Although we could add an interrupt capability we have no immediate plans to do so; since we know worst-case action execution delays (required for plan scheduling), any plan will contain only actions guaranteed to complete before any other plan execution deadlines pass. A prohibitively time-consuming action could never be part of a scheduled plan, so if such an action were absolutely required, CIRCA's AIS would simply fail.

8 Summary

Plans built for complex domains may not handle all possible states due to either imperfections in domain knowledge or approximations made to enhance planner efficiency. We have identified important classes of such unexpected situations, including "deadend", "removed", and "imminent-failure", and for each of these classes have described a means by which detection tests can be generated. When these tests detect an unhandled state, features are fed back to the planner, triggering replanning. We have implemented these algorithms in CIRCA and tested them in flight simulation. Tests demonstrated that the aircraft had a better chance to land safely when unhandled states were detected rather than being ignored.

CIRCA's reaction to unhandled states is coincidentally real-time, but this may not be acceptable when unhandled states quickly lead to failure. Timely reactions may be achieved either by bounding replanning and rescheduling execution times or by building reactions in advance. As planning technology progresses, more architectures employ methods for bounding planner execution ([Dean], [Ingrand], [Zilberstein]). We hope to enhance CIRCA's capabilities using such ideas.

9 References

E. M. Atkins, E. H. Durfee, and K. G. Shin, "Plan Development in CIRCA using Local Probabilistic Models," to appear in *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, August, 1996.

T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, "Planning with Deadlines in Stochastic Domains," *Proc. of AAAI,* pp. 574-579, July 1993.

M. L. Ginsberg, "Universal Planning: An (Almost) Universally Bad Idea," *AI Magazine*, vol. 10, no. 4, 1989.

F. F. Ingrand and M. P. Georgeff, "Managing Deliberation and Reasoning in Real-Time AI Systems," in *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, pp. 284-291, November 1990.

C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, January 1973.

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, pp. 83-127, 1995.

J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81-106, 1986.

R. Rainey, *ACM: The Aerial Combat Simulation for X11*. February 1994.

J. R. Rowland, *Linear Control Systems: Modeling, Analysis, and Design*, Wiley, 1986.

M. J. Schoppers, "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proc. Int'l Joint Conf. on Artificial Intelligence,* pp. 1039-1046, 1987.

D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley, "Planning and reacting in uncertain and dynamic environments," *Journal of Experimental and Theoretical AI*, vol. 7, no. 1, pp. 197-227, 1995.

S. Zilberstein, "Real-Time Robot Deliberation by Compilation and Monitoring of Anytime Algorithms," *AAAI Conferenc*e, pp. 799-809, 1994.