To appear in Working Notes of the AAAI Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control
Providence, Rhode Island, July 28, 1997

The CIRCA Model of Planning and Execution

Robert P. Goldman and **David J. Musliner** and **Mark S. Boddy** and **Kurt D. Krebsbach**
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{goldman,musliner,boddy,krebsbac}@htc.honeywell.com

Introduction

We are working to develop intelligent agents, using the CIRCA architecture (Musliner, Durfee, & Shin 1993; 1995), that construct and execute plans for controlling embedded real-time systems. By "embedded," we mean that these systems are interacting with a dynamic environment including uncontrolled, exogenous events. By "real-time," we mean that catastrophic failure is possible if a *timely* control action is not taken in certain situations. For these systems, control plans *must* provide guarantees that failures will not occur.

In this position paper we discuss our revised version of the CIRCA planning model. We start by reviewing the CIRCA architecture, which couples a deliberative planning component with a scheduler and a real-time executive. A distinctive feature of CIRCA is that it is based on a memoryless and unclocked reactive execution engine, but nevertheless manages to meet hard real-time constraints. A deliberative component of the architecture reasons about control actions, sensing actions, exogenous events and timing in order to achieve real-time behavior.

After reviewing the key features of CIRCA, we discuss a new planning technique, Dynamic Abstraction Planning (Goldman *et al.* 1997), that provides us with more efficient planning. We review the planning model that allows us to generate reactive controllers that provably meet real-time execution constraints. Finally, we discuss some ongoing extensions to the model. These extensions will allow CIRCA agents to take advantage of reliable processes in their environment, making them more capable controllers and allowing them to interact with other agents.

CIRCA

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

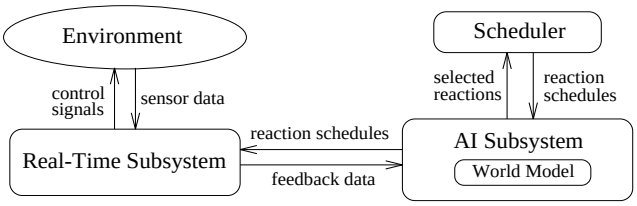

Figure 1: The Cooperative Intelligent Real-Time Control Architecture.

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics and simulated autonomous aircraft. In this paper we draw examples from a domain 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.

In this domain, CIRCA's planning and execution subsystems operate in parallel. The AIS 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 AIS is able to continue executing heuristic planning methods to find the next appropriate set of reactions. For example, the AIS 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, be-

```
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 2: Example transition descriptions given to CIRCA's planner.

cause 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 planning system builds reaction plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans (Musliner, Durfee, & Shin 1995). 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 2 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 AIS plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. The AIS 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. The assignment of actions determine 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 (Musliner, Durfee, & Shin 1995). 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 then 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. The test expression is a function of the plan as a whole, rather than local action assignments, because the same action may be assigned to more than one state. Some sample TAPs for the Puma domain are given in Figure 3. The `Max-per` indicates the maximum period at which the TAP must be run in order to meet the deadlines.

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 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 deadlines. These deadlines are captured as constraints on the maximum period of the TAPs (see Figure 3). This second phase of the translation process is done by the scheduler. In this phase, CIRCA's scheduler verifies that all actions in the TAP loop will be executed quickly enough to preempt the transitions 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. It is possible that scheduling will not succeed. In this case, the AIS will backtrack to the planner for the NFA to be revised, a new set of

```
#<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 3: Sample output from the TAP compiler.

TAPs generated and scheduled. The planning process is summarized in Figure 4.

Dynamic Abstraction Planning

The original CIRCA planner used a forward planning algorithm similar to that of Kabanza *et al.* (1997). This led to an explosion in the state space of the planner. In recent work (Goldman *et al.* 1997), we have addressed this state-space explosion using a new planning technique that we call Dynamic Abstraction Planning (DAP). Abstraction is used to omit detail from the state representation, reducing both the size of the state space that must be explored to produce a plan, and the size of the resulting plan itself. The abstraction method we describe has three useful features:

1. The abstraction method does not compromise safety-preserving guarantees: the world model used for planning is reduced, but not in ways that affect the system's ability to make rigorous statements about the safety assurances of plans it is building.
2. The method is fully automatic, and dynamically determines the appropriate level of abstraction during the planning process itself.
3. The method uses different levels of abstraction in different parts of the search space, individually adjusting how much detail is omitted at each step.

The intuition behind DAP is fairly simple: in some situations, certain world features are important, while in other situations those same features are not important. An optimal state space representation would capture

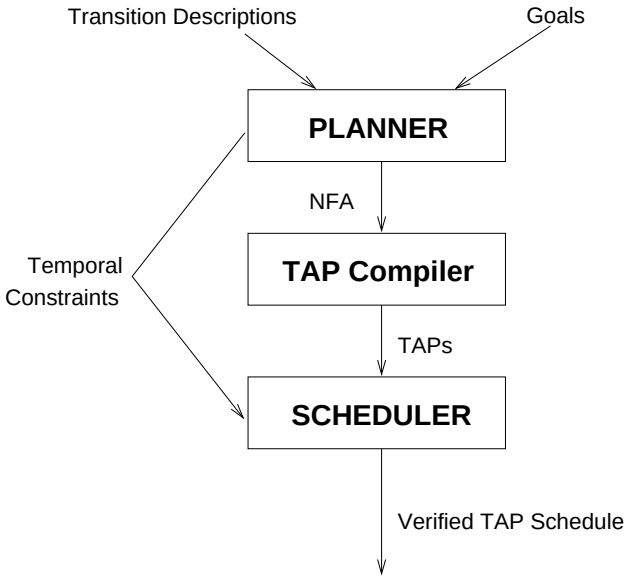

Figure 4: Summary of the CIRCA planning process.

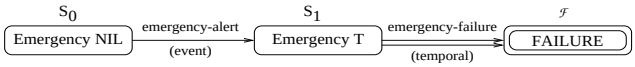

Figure 5: A partially-completed CIRCA plan.

only the important features for any particular state. In essence, DAP allows a planner to search for useful state space abstractions at the same time it is searching for a plan.

DAP Technique

Recall that the problem of planning for CIRCA (setting aside the question of TAP generation and scheduling), is to assign to every reachable state in the state space, an action that preserves safety. As mentioned above, the original CIRCA planner assigned these actions working forward from a set of start states. The CIRCA planner would maintain a frontier of reachable states and would assign a suitable action to each reachable state.

In contrast, the DAP planner begins with a very coarse NFA/plan, with all the non-failure states consolidated into a single state. DAP dynamically adds more detail to this sketchy NFA. DAP refines the NFA when it is unable to generate a satisfactory plan at the current level of detail. DAP refines the NFA by taking an existing state and splitting it into a number of more specific states, one for each possible value of a particular feature, F_i.

For example, let us consider the partially-completed plan given in Figure 5. Here there are three states: the failure state and two non-failure states, one for each value of `emergency`, a boolean proposition. This example is based on the domain model given in Figure 2. We assume that `emergency` is `nil` when the system

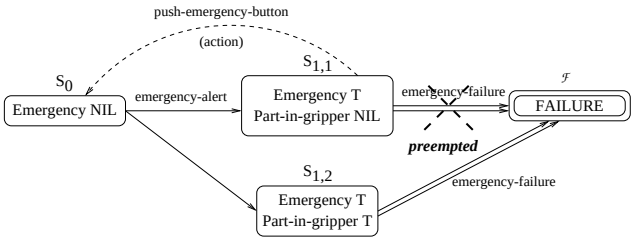

Figure 6: A refinement of the NFA in Figure 5.

begins operation.

The NFA in Figure 5 is not safe, because there is a reachable state, S_1, from which there is a transition to the failure state (`emergency-failure`) that has not been preempted. One way to fix this problem would be to choose an action for S_1 that will preempt `emergency-failure`. The domain description contains such an action, `push-emergency-button`. Unfortunately, one of `push-emergency-button`'s preconditions is `part-in-gripper= nil` and S_1 is not sufficiently detailed to specify values for `part-in-gripper`. We can rectify this omission by splitting S_1 into a set of states, one for each value of `part-in-gripper`. The resulting NFA is given in Figure 6. We can now assign `push-emergency-button` to solve the problem posed by state $S_{1,1}$. Further planning is required to resolve the problem posed by $S_{1,2}$, either by finding a preempting action that does not require `part-in-gripper` = `nil` or by making $S_{1,2}$ unreachable.

One unusual aspect of DAP is that detail is added to the NFA only *locally*. In our example above, we only added the feature `part-in-gripper` to the part of the state space where the `emergency` feature took on the value `true`, rather than refining all of the states of the NFA symmetrically. This introduces new nondeterminism: because we do not have a complete model of the initial state, we cannot say whether the `emergency-alert` transition will send the system to state $S_{1,1}$ or $S_{1,2}$.

Comparison to Other Abstraction Techniques

Many classical planning systems have used abstraction methods to increase the efficiency of searching for plans (see (Kambhampati 1994) for a brief survey). However, these abstractions are typically used only as guides in searching for a plan; the system may not know that its goals will actually be achieved by an abstract plan, and it will not be able to execute the abstracted operators directly. Instead, traditional abstraction planners must eventually expand their current plans down to the lowest level of detail, removing the abstraction to produce a final executable plan.

In the DAP approach, which involves abstraction only of state descriptions, abstract plans are executable, because the operators are always completely specified. This has two main advantages. First, the planning process can supply initial plans that preserve safety but might, on further refinement, do a better job of goal achievement. Second, the planning process can terminate with an executable abstract plan, which our results have shown may be much smaller than the corresponding plan expanded to precisely-defined states.

Dearden and Boutilier (1997) have developed an abstract planning algorithm for decision-theoretic planning modeled as a Markov decision process (MDP). Their method is similar to the DAP approach in that it involves aggregating states, but there are some differences. First, their method is not dynamic: aggregation is performed using a predefined set of "relevant" propositions, which is determined using Knoblock's approach (Knoblock 1994). Second, their method is uniform: the same propositions are relevant everywhere. The underlying model is also significantly different from CIRCA's: it does not model exogenous events or the timing required for real-time guarantees.

In previous work, Godefroid and Kabanza (1991) developed an abstraction technique based on partial orders. Their results allow a system to examine only a single ordering of independent actions, rather than enumerating all possible orderings. Unfortunately, these results are not immediately applicable to CIRCA, because their world model does not include exogenous events. The more recent work by Kabanza *et al.* (Kabanza, Barbeau, & St-Denis 1997) *does* include exogenous events, but they do not seem to have carried over the earlier abstraction concepts.

CIRCA Planning Model

The basic action model for CIRCA consists of *states*, each of which has an associated truth assignment over all the propositions in the domain of discourse, and *transitions*, which allow movement from state to state. Transitions can be partitioned on the basis of *volition*: in the current planner, *actions* are volitional, *events* and *temporals* are not. A transition is *enabled* in any state for which its *preconditions* are satisfied. The possible states resulting from a transition from a given state are those satisfying the transition's *postconditions*. The postconditions are specified, per the conventional STRIPS assumption, by listing only those literals that change values — other literals retain their values. The STRIPS assumption is loosened to a limited extent by permitting nondeterministic actions;[1] such actions have multiple sets of postconditions. The *delay* associated with a transition is known only in terms of upper

[1] There is no need to have nondeterministic events or temporals — for those it suffices simply to have multiple transitions.

and lower bounds.

A CIRCA *plan* is a graph, in which nodes correspond to sets of states and arcs represent possible transitions between those sets. A transition is *possibly* (respectively, *necessarily*) enabled if its preconditions are satisfied for some (all) element(s) of the set of states corresponding to the originating node, and its postconditions satisfied for some (all) element(s) of the set corresponding to the destination node. A plan in which all the node state sets have exactly one element corresponds to the original CIRCA planner (Musliner, Durfee, & Shin 1995).

A *well-formed* plan is one in which an arc is present for every possibly-enabled nonvolitional transition between nodes which is not *preempted*, and each node has outgoing arcs for at most one action (non-deterministic actions may require more than one arc). Preemption is defined below.

Timing information for a CIRCA plan is derived from bounds on the delay associated with arcs out of a node, which is taken directly from the delay bounds for the corresponding transitions. The *latency* of a transition arc with respect to a node in the plan is the time before that transition will occur, if no other transition occurs first, once some state in the set corresponding to that node has been reached. Latency bounds are path-dependent, which breaks the Markov assumption for nodes. We restore this property by calculating and employing path-independent bounds on latency in providing timing guarantees (most significantly, in determining preemption of transitions by actions).

Notation

- States: $s \in \mathcal{S}$.

  The set of states associated with a node in the plan graph we denote by $S \in 2^{\mathcal{S}}$. There being a $1:1$ relationship between sets S and nodes, we will use the set S to refer to the node.

- Transitions: $t \in \mathcal{T}$
  - $\mathsf{pre}(t)$ – preconditions of transition t
  - $S \models \Diamond t \equiv \exists s \in S, s \models \mathsf{pre}(t)$ – t is possibly enabled at node S
  - $d_{max}(t)$ – maximum delay for t
  - $d_{min}(t)$ – minimum delay for t

  The fact that latency bounds are calculated based on paths means that we must distinguish between arcs and the associated transitions.

- Arcs: $a \in \mathcal{A}$.
  - $\mathsf{tr}(a) \in \mathcal{T}$ – the transition label on a
  - $\mathsf{origin}(a) \in 2^{\mathcal{S}}$ – the node from which a leads
  - $\mathsf{result}(a) \in 2^{\mathcal{S}}$ – the resulting node.

  We will employ two syntactic substitutions:
  - $d_{min}(a)$, for $d_{min}(\mathsf{tr}(a))$ (also $d_{max}(a)$, *mutatis mutandis*)
  - $a \in (S, S')$, for $\mathsf{origin}(a) = S \wedge \mathsf{result}(a) = S'$

- Action assignments:
  - $\mathsf{act}(S)$ – the action assigned to state S

Transition timing

We assume that the "clock" for transition delay starts as soon as an enabling node (one in which $\mathsf{pre}(t)$ is possibly satisfied) is entered, and stops only when either the transition occurs, or a node is entered in which $\mathsf{pre}(t)$ is not possibly satisfied. In particular, the clock keeps running across other transitions between enabling nodes. This assumption applies to all transitions, volitional and nonvolitional.

Events and temporals For nonvolitional transitions (temporals and events), the bounds $[d_{min}, d_{max}]$ are specified as part of the domain description. In the current implementations, those bounds are:

- temporals: $[d, \infty]$
- events: $[0, \infty]$

Actions Actions are more complicated. The bounds on delay until an action happens are determined by the current TAP schedule, and by the delay associated with the action itself. One corollary of this statement is that the transition bounds for a given action are to a considerable extent determinable by the AIS.

Let's take a more detailed look at action timing. Under the control of a TAP schedule, the RTS takes a "snapshot" of the current state. It then evaluates that snapshot according to some test or sequence of tests, and decides whether or not to perform a given action. We assume complete and correct knowledge of the current state, so determination of what action to perform will be correct. The question is, how long will it take? There is a physical minimum time before the action could have an effect, consisting of the minimum time required for a test and the time required for the action itself. This is a "minimum upper bound" on the action (a lower bound on any specifiable d_{max}).

The precise execution of the TAP schedule is something we don't need to deal with at this point. For example, we neither need nor want to think about whether a single snapshot is tested for several actions, or whether each action takes its own snapshot. We assume that taking the snapshot, as opposed to testing, takes no time to accomplish. If this assumption

is relaxed, there's another scheduling optimization involved about when snapshots get taken and which tests are done on which snapshot. The characteristic that must be preserved is that the test for a given action is performed on a succession of snapshots, taken with no more than a specified maximum separation. The maximum delay before the action takes effect is then the sum of that maximum separation, the test delay, and the time required for the action itself.

d_{max} for actions is not an intrinsic feature, it's a parameter set by the planner in the planning process. Faced with a temporal transition to preempt, the planner can

- Choose an old action with a (previously specified) sufficiently small d_{max} for that node.
- Choose an old action with an insufficiently small d_{max} and specify a new, tighter bound.
- Choose a new action for the node and specify a sufficiently small d_{max}.
- Split the state, etc...

Note that any or all of these plan modifications may require a new TAP schedule to be generated. This suggests that the planner and scheduler should operate in fairly close synchronization. The current TAP schedule limits the allowable values for d_{max} for a given action (which specification in turn constrains the space of feasible schedules), while the suitability or otherwise of an action to preempt a given transition (what was previously called "applicability") is determined by that same specification.

Ghosting and inappropriate actions A further complication with actions is that the test and action are not atomic. It is entirely possible for some non-volitional transition to occur between the time that the current state is evaluated and the time the action takes effect. It is therefore possible for an action to be attempted in a state in which it is not technically "enabled."

The classical planning community calls these plans "ill-formed." For CIRCA, we adopt a similar convention, by defining the outcome of any such *inappropriate action* to be a failure state. Some inappropriate actions can be avoided by ensuring that the relevant (temporal) transitions are preempted. Events leading to unsuitable states cannot be preempted. This situation can be planned around, e.g. by splitting the node (separating the action and the event), splitting the event's destination (making the action be enabled in the result), choosing a different action, or declaring the current state a failure state as well.

Definitions

Preemption of one transition by another at a node is defined in terms of the latency bounds L_{min} and L_{max}:

$$\mathsf{preempts}(t, t', S) \equiv L_{max}(t, S) < L_{min}(t', S)$$

In words: t preempts t' in S iff t is guaranteed to occur before t' once S is reached, no matter how you got there.

The *maximum dwell* of a node S is relevant because we can guarantee that no transition out of that node will take place with a longer delay.

$$D_{max}(S) = \min_{a \in (S,X)} L_{max}(\mathsf{tr}(a), S)$$

The lower bound on latency for a transition t at a node S is the lower bound on delay for t, unless there are "enabling predecessors" (defined below), in which case the lower bound on latency is the minimum value derivable from those predecessors.
if $\mathsf{ep}(t, S) = \emptyset$ then

$$L_{min}(t, S) = d_{min}(t)$$

else

$$L_{min}(t, S) = \min_{S' \in \mathsf{ep}(t,S)} L^*_{min}(t, S, S')$$

The lower bound on latency for S derivable from an enabling predecessor S' is recursively defined as $L_{min}(t, S')$, minus the maximum possible transition time from S' to S.

$$L^*_{min}(t, S, S') = L_{min}(t, S') - D^*_{max}(S')$$

where

$$D^*_{max}(S') = \min D_{max}(S'), \max_{\mathsf{othert}(t,S',S)} L_{max}(\mathsf{tr}(a), S')$$

and

$$\mathsf{othert}(t, S', S) = \{a \in (S', S) | \mathsf{tr}(a) \neq t\}$$

An *enabling predecessor* (ep) for t at S is any node S' at which t is enabled, from which S is reachable by an arc with some label *other than* t (otherwise the clock resets).

$$\mathsf{ep}(t, S) = \{S' | S' \models \Diamond t \wedge \exists a \in (S', S), \mathsf{tr}(a) \neq t\}$$

The upper bound on latency for t at S is the maximum delay $d_{max}(t)$, unless there are enabling predecessors, *et cetera*.
if $\forall a, \mathsf{result}(a) = S \Rightarrow \mathsf{origin}(a) \in \mathsf{ep}(S)$ then

$$L_{max}(t, S) = \max_{S' \in \mathsf{ep}(t,S)} L^*_{max}(t, S, S')$$

otherwise

$$L_{max}(t, S) = d_{max}(t)$$

The upper bound on latency for S derivable from an enabling predecessor S' is recursively defined as $L_{max}(S')$ minus the minimum possible transition time from S to S'.

$$L^*_{max}(t, S, S') = L_{max}(t, S') - \min_{\mathsf{othert}(t,S',S)} L_{min}(\mathsf{tr}(a), S')$$

One of the interesting results of this timing model is that one can achieve "better than real-time" performance. Given a node with a troublesome temporal, say one where L_{min} is less than any achievable d_{max} for the desired action(s), preemption can be guaranteed by ensuring that the node is only reachable from nodes at which the action is enabled, and only via temporal transitions with a sufficiently large L_{min}. The current planner does not exploit this opportunity, and we have no immediate plans to do so.

There are several simplifications we can make. We start by assuming that $L_{max}(t, S) = d_{max}(t)$ in all cases. This assumption preserves the correctness of the latency bounds and preemption calculations, by virtue of the fact that $L_{max}(t, S) \leq d_{max}(t)$. The bound is weaker only in the somewhat peculiar "better-than-real-time" case described above.

This leads to additional simplifications. Here is the complete set of revised definitions. By assumption:

$$L_{max}(t, S) = d_{max}(t)$$

For the maximum dwell, we use the assumption above, plus the fact that there is exactly one action specified for an node in the plan graph ($d_{max}(\text{no-op}) = \infty$):

$$D_{max}(S) = d_{max}(\mathsf{act}(S))$$

L_{min} does not change. However, L^*_{min} does:

$$L^*_{min}(t, S, S') = L_{min}(t, S') - d_{max}(\mathsf{act}(S'))$$

It doesn't matter whether $\mathsf{result}(\mathsf{act}(S')) = S$ or not. Also note that the definition of ep has not changed.

Algorithm

Calculating L_{max} and D_{max} is reduced to lookup operations. The simplified definition of L^*_{min} above suggests a simple depth-first graph search, from nodes to their enabling predecessors. The algorithm has an additional termination condition: terminate with a bound of zero any time the summed "path cost" (d_{max} values for the appropriate actions) is greater than $d_{min}(t)$. This termination condition allows this algorithm to complete even in plans (graphs) with cycles: once the computed L_{min} along any path drops to (or below) 0, we're done.

Related Work

Kabanza *et al.* (Kabanza, Barbeau, & St-Denis 1997) have developed a planning method for reactive agents that is similar to the original CIRCA. Their architecture differs in emphasis, however. The NFAs it constructs are "clocked:" they make transitions at times that are the least common denominator of all possible transitions. This scheme will suffer a state space explosion in domains where there is a wide range of possible transition delays, like those to which CIRCA has been applied. Kabanza's group has concentrated on developing a more flexible notation for goals than those used by CIRCA, but they do not make the same distinction between safety and goal achievement.

Expanding Expressiveness

At the same time that we have been working to increase the efficiency of CIRCA's planning, we are working to relax limits on its expressiveness. In doing this we have been driven by consideration of the scenario outlined by Gat in his paper "News From the Trenches: An Overview of Unmanned Spacecraft for AI" (Gat 1996).

In this paper, Gat presented a planning scenario from the Cassini mission that he argued no current AI planning system could tackle. The problem concerns the Saturn orbital insertion of the Cassini spacecraft. In order to successfully navigate, the Cassini spacecraft must have an inertial reference unit (IRU) powered up and functioning. The spacecraft has a primary and a secondary IRU. The problem is to foresee the possibility of a primary IRU failure and warm up both IRUs early enough that they will be available for navigation at the time of orbital insertion.

CIRCA is quite capable of planning to warm up both IRUs, provided that it is informed that doing orbital insertion without guidance is a failure and that the primary IRU can fail. CIRCA can do this because, unlike most other planners, [2] CIRCA considers and plans against, external events. CIRCA can warm up the IRUs early enough, because of its temporal reasoning.

However, this scenario has led us to consider two shortcomings of the current CIRCA approach. First, CIRCA considers exogenous processes only as threats, rather than as opportunities. CIRCA's planner only chooses either to preempt exogenous processes or allow them to happen. Accordingly, the current CIRCA

[2] With the exception of Blythe's (1996) and Kabanza's (Kabanza, Barbeau, & St-Denis 1997)

world model provides only lower bounds on the delay of temporal transitions. This makes it impossible for CIRCA to rely on external processes (like the warming of an IRU), because doing so requires CIRCA to reason about the *upper bound* on the duration of the warming process.

A second shortcoming has to do with the lack of a systemwide clock. Currently, CIRCA can reason only about duration relative to the time it enters a particular state. In order to properly meet deadlines, as in this example, where the IRUs must be warmed prior to orbital insertion, the RTS must be able to act at an appropriate time relative to a planned future event.

We have developed preliminary solutions to the above two problems. The existing temporal model already takes into account some upper bounds — those on the duration of actions. We plan to expand the model to include *reliable temporals*, with upper bounds on their time of completion, together with state-encoding of the progress of those processes.

We are also addressing the problem of CIRCA not having a systemwide clock. We do not want to abandon the unclocked executive, because inclusion of global time into the state space can cause it to explode (see comparison to Kabanza's execution model earlier). What we would like to do is to provide chosen clock signals for particular times to the RTS. It is certainly possible to provide such signals — for most applications like autonomous spacecraft, there will be a system or mission clock. What we need to be able to do is to identify important times and set up signals to the RTS accordingly. The RTS will then detect these signals like any other state feature. Our preliminary investigations suggest that we can detect the need for such features through search failures in the AIS.

Overcoming these expressive limitations is an important area of ongoing theoretical investigation at HTC. We hope to begin experimenting with solutions to these problems sometime this year.

Conclusions

In this paper we have presented our approach to achieving intelligent, real-time performance. This approach is based around the coupling of a deliberative system with a memoryless, unclocked real-time reactive execution module. Through its planning model, the system is able to achieve real-time behavior without incorporating a representation of time in its execution engine. We have discussed ways of efficiently manipulating this model through dynamic abstraction. Finally, we introduced current work on relaxing some of the limitations imposed by the current planning model.

Acknowledgments This work was supported by the Defense Advanced Research Projects Agency under contract DAAK60-94-C-0040-P0006.

References

Blythe, J. 1996. A representation for efficient planning in dynamic domains with external events. in the AAAI workshop on "Theories of Action, Planning and Control: Bridging the gap".

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1–2):219–283.

Gat, E. 1996. News from the trenches: An overview of unmanned spacecraft for AI. In Nourbakhsh, I., ed., *AAAI Technical Report SSS-96-04: Planning with Incomplete Information for Robot Problems*. American Association for Artificial Intelligence. Available at `http://www-aig.jpl.nasa.gov/home/gat/gp.html`.

Godefroid, P., and Kabanza, F. 1991. An efficient reactive planner for synthesizing reactive plans. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 640–645. Cambridge, MA: MIT Press.

Goldman, R. P.; Musliner, D. J.; Krebsbach, K. D.; and Boddy, M. S. 1997. Dynamic abstraction planning. To appear in the proceedings of the 1997 National Conference on Artificial Intelligence.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning control rules for reactive agents. Technical Report 197, Computer Science Dept., University of Sherbrooke.

Kambhampati, S. 1994. Refinement search as a unifying framework for analyzing planning algorithms. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Principles of Knowledge Representation and Reasoning:Proceedings of the Fourth International Conference*. Morgan Kaufmann Publishers, Inc.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68:243–302.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1561–1574.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.