Appears in Working Notes of AAAI Symposium on Hybrid Control
Stanford; March, 1999

Hard Real-time Mode Logic Synthesis for Hybrid Control
A CIRCA-based approach

Robert P. Goldman, Mike Pelican, David J. Musliner
{ goldman,pelican,musliner}@htc.honeywell.com

Introduction

We are developing intelligent, autonomous, flexible control systems for mission-critical applications such as Uninhabited Aerial Vehicles (UAVs) and deep space probes. These applications require hybrid control systems, capable of effectively managing both discrete and continuous controllable parameters to maintain system safety and achieve system goals. This report describes our work on developing intelligent, hard real-time controllers for such hybrid systems. In particular, we focus on the techniques we are developing for automatically synthesizing guaranteed mode-switching controllers from domain models.

The mission-critical systems we are interested in are typically designed, built, verified, and fielded by very large organizations such as defense contractors. The overall system control problems for these products are not solvable "in the whole" because of sheer magnitude, theoretical limitations, and because they require the integration of specialists from many different disciplines. Instead, the overall control problems are decomposed into multiple simpler control problems. For example, in an aircraft controller, pitch, roll and yaw are typically done separately and indeed, often by different avionics firms.[1] Even a relatively primitive behavior such as coordinated turn is several layers up the control hierarchy. Autopilots provide sets of qualitative modes, such as "Altitude hold," "Altitude select," "Vertical speed," etc.

Piecing together the different controllers, the system's *mode logic* is both critical and extremely complex. We generate this kind of logic, in the form of clocked, discrete controllers. These controllers do *not* address the continuous part of the problem (except in the temporal dimension). Instead, the behavior of individual control modes is summarized in terms of a *qualitative* feature space and in terms of bounds on the timing of transitions between qualitative states. Note that the kind of mode logic that we are discussing is not simply sequencing a single control system through multiple modes; it also covers sequencing multiple control systems in coordination. To take a simple example, the developer of an autopilot module that is designed to carry an aircraft from one waypoint to another typically does not concern him/herself with handling the behavior of the aircraft while its landing gear are deployed. Accordingly, it will be the job of our "outermost-loop" discrete control to ensure that the landing gear are *not* deployed when the autopilot is switched to this mode.

In our approach, these controllers are synthesized *automatically* and dynamically, on-line, *while the platform is operating.* Dynamic generation allows us to handle both the familiar problem of state space explosion, and a related problem we call *bounded reactivity* (Musliner, Durfee, & Shin 1993): even simple, reactive systems have only limited ability to monitor their environments and act. If too many contingencies must be addressed, the system will not be able to react in a timely way. Dynamic generation of mode-logic allows our system to tailor its reactive subsystem to the immediately relevant contingencies. Rather than requiring the system to monitor all contingencies over an entire mission, we develop a set of specialized discrete state controllers for different parts of the mission.

In this paper, we discuss current work on adapting the CIRCA architecture (Musliner, Durfee, & Shin 1993; 1995) to control hybrid systems. We have augmented the knowledge representation used in developing CIRCA's reactive plans (real-time discrete event controllers) to be better suited to developing mode logic. To provide the temporal reasoning needed to use this knowledge, we have incorporated a model-checker into CIRCA's Controller Synthesis Module. We are in the process of adding an Adaptive Mission Planner and a monitoring subsystem to the architecture. The Adaptive Mission Planner will provide long-term

[1] "The current allocation of flight automation to separate functions is the result of largely accidental historical factors. Consequently, certain control variables that are tightly coupled in a dynamical sense are managed by different functions: for example, engine thrust is managed by the autothrottle, and pitch angle by the autopilot." (Rushby 1998)

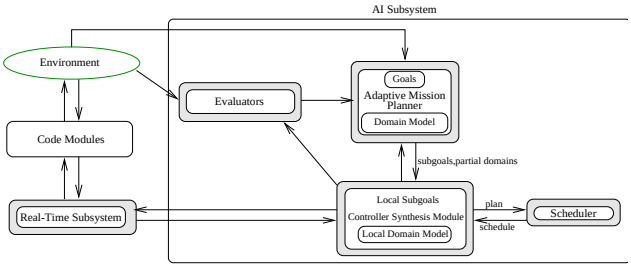

Figure 1: The CIRCA architecture as recently expanded.

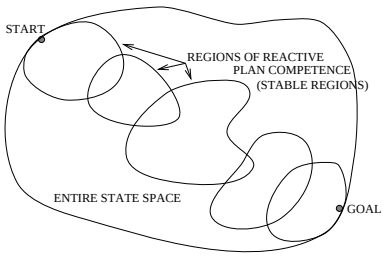

Figure 2: Conceptual view of multiple reactive controllers.

(mission duration) projection and reasoning about resources and decompose the problem space for the benefit of the Controller Synthesis Module. The monitoring subsystem will monitor progress towards goals, alerting the mission planner when the system is not moving towards its objectives quickly enough.

CIRCA

CIRCA has been applied to real-time planning and control problems in several domains including mobile robotics, simulated autonomous aircraft, space probe challenge problems (Musliner & Goldman 1997) and controlling a fixed-wing model aircraft (Atkins *et al.* 1998). CIRCA is designed to support both hard real-time response guarantees and unrestricted (i.e., intractable) AI methods that can guide those real-time responses. In CIRCA, 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.

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 (a discrete controller). While the RTS is executing those reactions, ensuring that the system avoids failure, the AIS continues using heuristic planning methods to find the next appropriate set of reactions. For example, while the RTS is executing a controller that will take a UAV from its start point through a sequence of waypoints to its target, the AIS will be generating a new controller for the UAV's actions over the target (e.g., overflying a particular location and photographing it). When this controller has been generated and the UAV has reached an appropriate position, the new schedule of reactions will be downloaded to the RTS.

A diagram of the current CIRCA system architecture is given in Figure 1. In our current system, the AIS is partitioned into an Adaptive Mission Planner (AMP) a Controller Synthesis Module (CSM) and a scheduler. The AMP reasons about mission profiles in a quasi-linear way, carving the overall mission control plan into smaller regions that are fed to the CSM(see Figure 2). For each of these control regions, the CSM generates a timed discrete event controller that will assure system safety and attempt to achieve system goals. Put differently, the AMP divides the overall mission state space into a series of smaller state spaces, for which the CSM generates controllers.

The AMP also drives the generation of *evaluators*. These are used to give feedback to the AMP as a mission unfolds. The AMP reasons about overall mission achievement, while the running set of reactions are responsible only for moving towards *local* subgoals (region intersections in Figure 2). The evaluators provide the AMP with a constantly-updated view of the current world state, watching both the progress of the agent towards its objective and the planning activity of the CSM. Space will not permit detailed discussion of the monitoring subsystem here.

CIRCA's Controller Synthesis Module 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. In the existing system, these transitions are of four types:

Action transitions represent actions performed by the RTS.

Temporal transitions represent the progression of time and continuous processes that may need to be preempted.

Event transitions represent world occurrences as instantaneous state changes.

Reliable temporal transitions represent continuous processes (such as the operation of a control law) that may need to be employed by the CIRCA agent.

```
;; the action of switching on an Inertial
;;Reference Unit (IRU)
ACTION start_IRU1_warm_up
   PRECONDITIONS: '((IRU1 off))
   POSTCONDITIONS: '((IRU1 warming))
   DELAY: <= 1

;; the process of the IRU warming
RELIABLE-TEMPORAL warm_up_IRU1
   PRECONDITIONS: '((IRU1 warming))
   POSTCONDITIONS: '((IRU1 on))
   DELAY: [45 90]

;;sometimes the IRUs break without warning
EVENT IRU1_fails
   PRECONDITIONS: '((IRU1 on))
   POSTCONDITIONS: '((IRU1 broken))

;; if the engine is burning while the active
;; IRU breaks, we have a limited amount of
;; time to fix the problem before the
;; spacecraft will go too far out of control
TEMPORAL fail_if_burn_with_broken_IRU1
   PRECONDITIONS: '((engine on)(active_IRU IRU1)
                    (IRU1 broken))
   POSTCONDITIONS: '((failure T))
   DELAY: >= 5
```

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

For example, Figure 3 shows several transitions used in a situation where CIRCA is to control the Cassini spacecraft in Saturn Orbital Insertion.[2] Note that these transition descriptions implicitly provide a factored description of a timed discrete controller state space.

The CSM plans by generating a timed nondeterministic finite automaton (NFA) (Alur 1998) from these transition descriptions. It assigns to each reachable state either an action transition, a reliable temporal or `no-op`. These selections are made to *preempt* transitions that lead to failure states and to drive the system towards goal states. A transition, t, is preempted in a state when that state is assigned an action or reliable temporal that is *guaranteed* to occur before t can possibly occur.

The assignment of actions and reliable temporals 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 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 compiled 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). Each TAP has a boolean test expression that distinguishes 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.

These TAPs are then assembled into a loop that will meet all the deadlines. These deadlines are captured as constraints on the maximum temporal separation of the TAPs. This second phase of the translation process is done by the scheduler. CIRCA's scheduler verifies that all actions in the TAP loop will be executed quickly enough to preempt the transitions the CSM 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 CSM to revise the NFA and generate and schedule a new set of TAPs.

When the TAPs are arranged into an executable loop, they 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.

For example, in a recent test of the CIRCA RTS, it was used to pilot two simulated UAVs in an attack mission. Each UAV's RTS controller, following a mission profile, first triggers a control subsystem that causes the aircraft to take off. It then switches the autopilot into a waypoint-directed flight mode. While the aircraft are flying from one waypoint to the next, the RTS will be running a number of TAPs in rotation. Some important TAPs are:

- One that checks to see whether the aircraft has reached the next waypoint and, if so, sequences the next waypoint.
- One that checks to see if the aircraft has been "painted" by enemy radar and, if so, takes countermeasures (in this case releasing chaff).
- One that checks to see if the aircraft is in the vicinity of a valid ground target and is currently the LEAD. If so, the aircraft will attempt to destroy the target.

[2] The problem is taken from Erann Gat's "From the Trenches" (Gat 1996).

Generating Mode-logic for Hybrid Control

We are revising the CIRCA controller synthesis algorithm to be more suitable for generating mode logic for hybrid systems. The new CIRCA CSM employs an abstraction technique that we call "dynamic abstraction planning" (DAP) (Goldman *et al.* 1997) to help control the state space explosion. We have also revised the CSM algorithm to incorporate a real-time model-checker. Finally, we have augmented the CSM's knowledge representation, complicating the controller synthesis problem.

In the original CIRCA, which drove a mobile robot, the RTS was intended to directly interact with the plant: RTS tests would sample sensors and RTS actions would control effectors. This simple discrete-event control approach is not suitable for mode logic in which the RTS must control other, lower-level controllers. Instead, the RTS must use temporally-extended processes, both those in the environment itself and those that are created by subsidiary controllers. To do this, we have introduced the *reliable temporal* class of transition referred to in the previous section.

DAP Technique

CIRCA generates its controllers through a process that we call "dynamic abstraction planning" (DAP) (Goldman *et al.* 1997). 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, adjusting the amount of detail omitted at each step.

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

The controller synthesis problem for CIRCA is to assign to every reachable state in the state space an action or reliable temporal that preserves safety, by preempting any *possibly* applicable transitions to a failure state. In order to assign an action to a state, the action must *necessarily* apply. The DAP planner begins with a very coarse NFA/plan, with all the non-failure states consolidated into a single state. In a state with so few features defined, nearly all transitions to failure are possibly applicable and almost none of the actions are necessarily applicable.

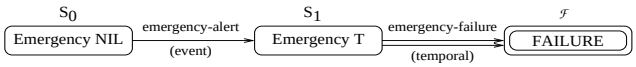

Figure 4: A partially-completed CIRCA plan.

During the planning process, DAP dynamically adds more detail to this sketchy 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. Features are chosen for splitting because they either make a potential failure state unreachable (by making a possibly applicable transition to failure inapplicable) or enable a preempting action choice (by making a possibly applicable action necessarily applicable).

For example, let us consider the partially-completed plan given in Figure 4. Here there are three states: the failure state and two non-failure states, one for each value of `emergency`, a boolean proposition. We assume that `emergency` is `nil` when the system begins operation. The NFA in Figure 4 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`. We can now assign `push-emergency-button` to solve the problem posed by state $S_{1,1}$. The resulting NFA is given in Figure 5. 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 `t`, rather than refining all of the states of the NFA symmetrically.

The CIRCA CSM algorithm is summarized in Figure 6. The algorithm is presented as a non-

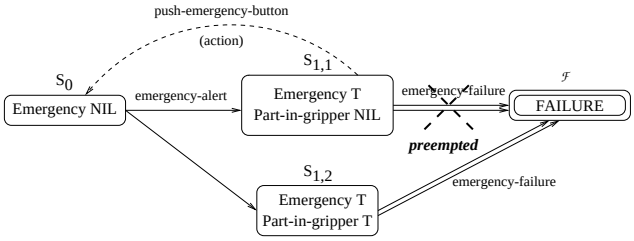

Figure 5: A refinement of the NFA in Figure 4.

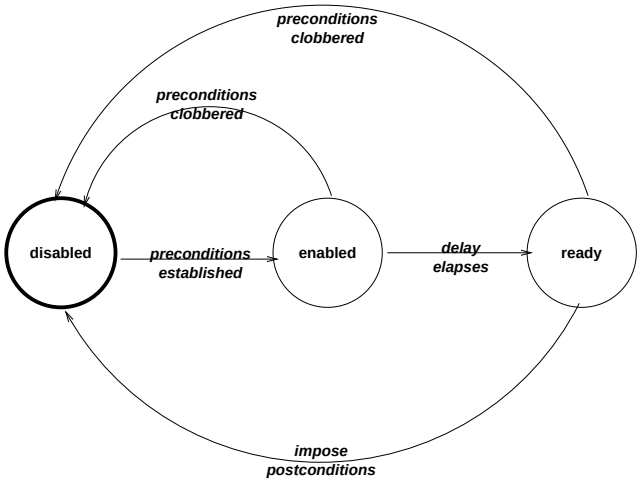

Figure 7: Clocked automaton representing the states of a temporal transition.

```
abstract-plan (isd);
isd is initial state description
let N = ∅; The graph
    openlist = ∅;
    is = make-initial-state(isd);
N := N ∪ {is};
push(is, openlist);
loop
    if there are no more reachable states in the openlist then
        we are done
        break;
    else
        let s = choose a reachable state from openlist;
            openlist := openlist − {s};
            oneof
                split-state :
                    choose a proposition p and split s into |val(p)|
                            states;
                    remove s from N and insert the new states;
                    add the new states to the open list;
                assign-action :
                    choose an action applicable to s;
                    (†) verify timing properties of partial plan, N
                utilize-process :
                    choose a reliable-temporal applicable to s;
                    (†) verify timing properties of partial plan, N
                fail
```

Figure 6: Pseudo-code for the CIRCA state-space planning algorithm.

deterministic algorithm, with choice points marked in bold-face. The algorithm is implemented by search, using intelligent backjumping.

Verification of Intermediate Plans

In the discussion above, we have ignored the temporal aspects of CIRCA state space planning. In particular, we have drawn the plan/controller as if it were a simple nondeterministic finite automaton (NFA). However, the controller is in fact the *product* of a set of *clocked* NFAs and should be analyzed accordingly (Alur 1998). To do so requires that we reason about multiple paths through the automaton, because the temporal aspect of the automaton is not Markovian in a useful sense. To do that reasoning, we use a real-time model-checker.

One can view the overall CIRCA plan as being the product of the following automata:

- one automaton for each temporal and reliable temporal,
- one automaton representing action choice, and
- one automaton for the execution of each action assigned to a state.

For example, Figure 7 shows the states a temporal transition may be in: initially it is disabled. When the system enters a state in which the temporal's preconditions are satisfied, the transition becomes enabled. At this point, one of two things can happen: (1) the preconditions remain enabled, in which case, after the (lower bound) delay has elapsed, the machine moves to the ready state; or (2) the preconditions get clobbered and the machine returns to disabled. In the

ready state, at any time the transition may fire, asserting its postconditions and returning to disabled or, if the preconditions are denied, the machine will simply return to the disabled state.

The automata representing action execution are similar but slightly complicated by the need to consider the "sense-act gap" — the delay between the moment when the sensory snapshot was taken and the chosen action actually occurs.

Note that temporal transitions can remain enabled through a number of states in the overall NFA (the product machine). This complicates the temporal reasoning needed to verify correct timing. To solve this problem, we have used the KRONOS (Yovine 1998) model-checking tool to verify the timing of (partial) plans. When an action or reliable temporal is assigned to preempt a temporal transition, t, the DAP algorithm uses KRONOS to verify that t cannot occur in the planned-for region of the state space (see (†) in Figure 6). In the event that the timing requirements are not met (i.e., t can occur), KRONOS returns a path through the state space that leads to this failure. DAP then uses the path to failure to identify the planning decisions that led to failure, and to select one as a backjumping target.

Backjumping

DAP's ability to identify culprit decisions and backjump has proven critical to its ability to find plans within a reasonable amount of time. The CIRCA CSM search space has two characteristics that make it very difficult to search using a naive chronological backtracking approach. First, the space is very large. Second, it is very difficult to identify when a decision made early in the search process has made a solution infeasible. Typically, it is only when a sequence of actions and temporal transitions have been specified that the failure condition is realized.

We have used a dependency-directed backjumping approach to overcome these difficulties. When the KRONOS verifier determines that failure is reachable in an intermediate plan, it finds path from an initial state to a failure state and returns it to the planner. From that path, a list of implicated decisions is extracted and the planner backjumps to the most recent of those decisions.

The backjumping is complicated by the use of reliable temporals as a way of employing domain processes and special-purpose controllers. Essentially, the problem of using a reliable temporal is one of constructing a sufficiently long chain of states in which that reliable temporal is enabled. The difficulty is to augment the backjumping with enough information about how to correct problems in such a chain while not making the search incomplete. Efforts to overcome this challenges are a focus of current research.

Related Work

Our work on controller synthesis is similar in its subject to the controller synthesis work of Maler, Pneuli, and Sifakis (1995) (henceforth MPS). MPS have developed a game-theoretic framework for synthesizing discrete controllers for timed systems[3] and show that the controller synthesis problem for these systems is decidable. More recently, Tomlin, Lygeros, and Sastry (1998) have extended the MPS method to the synthesis of hybrid automata, even in the presence of nonlinear continuous dynamics. Unlike MPS, we are interested in *completely automated, on-line* controller synthesis, so computational efficiency is critically important. To simplify the problem, we have limited the power of our controllers, and carefully limited the size of the state space using the AMP, factoring the state space, and using the DAP method. We also provide an execution platform for our controllers (the RTS) and a scheduling theory.

In the AI literature, Kabanza, Barbeau, and St.-Denis (KBS) (1997) also work on generating timed discrete controllers. KBS's work differs in assuming a discrete time model, and representing time explicitly in the controller. Our controllers are unclocked (although they meet timing constraints), making them less powerful but easier to synthesize; representing time explicitly in the controller can lead to a state space explosion. Recent research based on Markov Decision Processes (e.g., (Dearden & Boutilier 1997)), differs from ours in considering uncertainty to be more important than timing. For the control problems that interest us, a simple representation of uncertainty is adequate, while correct handling of timing is critical.

Conclusions and Future Work

We argue that, for many autonomous hybrid systems, the key to effective, intelligent control, is the engineering of appropriate mode logic. Further, we argue that this mode logic can be effectively generated by model-based reasoning from the temporal and discrete event features of special-purpose controllers. We have developed an algorithm for generating mode-logic automatically from such descriptions and verifying its timing properties through the use of a model-checker. We are currently applying the CIRCA techniques described in this paper to controlling a simulated combat UAV, under support from DARPA's Active Software Composition program.

[3] Their "forcing a win" parallels our "necessarily avoiding failure."

Acknowledgments This work was supported by the Defense Advanced Research Projects Agency under contract F30602-98-C-0212.

References

Alur, R. 1998. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*.

Atkins, E. M.; Miller, R. H.; VanPelt, T.; Shaw, K. D.; Ribbens, W. B.; Washabaugh, P. D.; and Bernstein, D. S. 1998. Solus: An autonomous aircraft for flight control and trajectory planning research. In *Proceedings of the American Control Conference (ACC)*, volume 2, 689–693.

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`.

Goldman, R. P.; Musliner, D. J.; Krebsbach, K. D.; and Boddy, M. S. 1997. Dynamic abstraction planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 680–686. Menlo Park, CA: American Association for Artificial Intelligence.

Kabanza, F.; Barbeau, M.; and St.-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Maler, O.; Pneuli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In Mayr, E. W., and Puech, C., eds., *STACS 95: Theoretical Aspects of Computer Science*. Springer Verlag. 229–242.

Musliner, D. J., and Goldman, R. P. 1997. CIRCA and the Cassini Saturn orbit insertion: Solving a prepositioning problem. In *Working Notes of the NASA Workshop on Planning and Scheduling for Space*.

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.

Rushby, J. 1998. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. draft version.

Tomlin, C.; Lygeros, J.; and Sastry, S. 1998. Synthesizing controllers for nonlinear hybrid systems. In *Lecture Notes in Computer Science 1386, Proceedings of Hybrid Systems: Computation and Control*. Springer Verlag.

Yovine, S. 1998. Model-checking timed automata. In Rozenberg, G., and Vaandrager, F., eds., *Embedded Systems*. Springer Verlag.