Appears in Working Notes of the AAAI Spring Symposium on Real-Time Autonomous Systems
Stanford, CA, March 2000

Modeling and Verification for Automatic Synthesis of Real-time Controllers

Robert P. Goldman, Michael J. Pelican, David J. Musliner
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{goldman, pelican, musliner}@htc.honeywell.com

Introduction

We have developed a novel technique for automatically synthesizing hard real-time reactive controllers using model-checking verification. Our algorithm builds a controller incrementally, using a timed automaton model to check each partial controller for correctness. The verification model captures both the controller design and the semantics of its execution environment. If the controller is found to be incorrect, information from the verification system is used to direct the search for improvements. This paper describes how our controller synthesis process uses verification, and explains in detail how we model the execution of the real time subsystem of the CIRCA intelligent control architecture.

We are developing autonomous, flexible control systems for mission-critical applications such as Unmanned Aerial Vehicles (UAVs) and deep space probes. These applications require hybrid real-time control systems, capable of effectively managing both discrete and continuous controllable parameters to maintain system safety and achieve system goals. Using the CIRCA architecture for adaptive real-time control systems (Musliner, Durfee, & Shin 1993; 1995; Musliner *et al.* 1999), these controllers are synthesized *automatically* and dynamically, on-line, *while the platform is operating.* Unlike many other intelligent control systems, CIRCA's automatically-generated control plans have strong temporal semantics and provide safety guarantees, ensuring that the controlled system will avoid all forms of mission-critical failure.

CIRCA uses model-checking techniques for timed automata (Alur 1998; Yovine 1998) as an integral part of its controller synthesis algorithm. CIRCA's *Controller Synthesis Module*
(CSM) incrementally builds a hard real time reactive controller from a description of the processes in its environment, the control actions available and a set of goal states. To do this, the Controller Synthesis Module must build a model of the controller it is constructing that is faithful to its execution semantics, and use this model to verify that the controller will function safely in its environment.

In the following section, we give a brief overview of the structure and purpose of the CIRCA architecture. Then we briefly describe the CSM module and its synthesis algorithm, wrapping up with a discussion of the way the CSM uses a timed-automaton verifier. After this, we describe the modeling of the CIRCA real time executive as a set of interacting timed automata. Finally, we conclude with some comparison to related work and some mention of future research directions.

CIRCA

The CIRCA architecture is intended to provide intelligent control to autonomously-operating systems.[1] To do this, CIRCA must operate at multiple time scales. CIRCA must be able to reason about the profile of a mission as a whole. For example, if CIRCA is operating an Uninhabited Combat Aerial Vehicle (UCAV), its mission-level planning must be able to reason about issues like fuel use and navigation to its goal. At a lower level, CIRCA must have a controller that is able to react to threats and opportunities that arise in its immediate environment. For example, when targeted by enemy radar, the CIRCA-controlled UCAV must carry out countermeasures (e.g., release chaff) and initiate evasive maneuvers. Furthermore, CIRCA must guarantee that these reactions will be taken in time. It is not enough to *eventually* release chaff; CIRCA must inspect its environments for threats sufficiently often, and must react to those threats within specified time bounds.

[1]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).

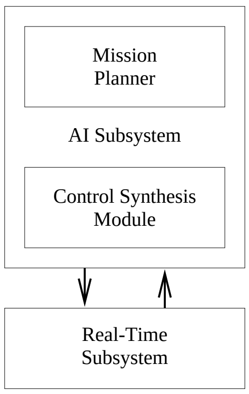

Figure 1: Basic CIRCA architecture.

CIRCA employs two strategies to manage this complex task. First, its mission planner decomposes the mission into more manageable subtasks that can be planned in detail. Second, CIRCA itself is decomposed into two concurrently-operating subsystems (see Figure 1): an *AI Subsystem* (AIS) reasons about high-level problems that require powerful but potentially unbounded computation, while a separate *real-time subsystem* (RTS) reactively executes the AIS-generated plans and enforces guaranteed response times. The AIS contains the CSM, which is the focus of this paper, as well as the mission planner and some support modules, none of which we will discuss here.

The Controller Synthesis Module (CSM) bridges mission-level planning and reactive control. It takes descriptions of a phase of a system mission and dynamically, automatically, synthesizes a set of reactions that maintain the system's safety and move it towards its goals. When this controller is operating, the CSM will be working to generate controllers for other phases of the mission.

The Controller Synthesis Module

The objective of the CIRCA CSM is to automatically synthesize hard real-time discrete controllers that guarantee system safety when run on CIRCA's real-time subsystem. The CIRCA CSM builds reactive discrete controllers that observe the system state and some features of its environment and take appropriate control actions. In constructing such a controller, the CSM takes a description of the processes in the system's environment, represented as a set of transitions that modify world features and that have worst case time characteristics. From this description, CIRCA incrementally constructs a set of reactions and checks them for correctness using a timed automaton verifier.

CIRCA's reactive controllers

The real-time controllers that CIRCA builds sense features of the system's state (both internal and external), and execute reactions based on the current state. That is, the CIRCA RTS runs a *memoryless* reactive controller. Note particularly, that CIRCA does not maintain any internal clocks, so time is not a feature used in choosing control actions. The CIRCA system achieves performance guarantees by analyzing the execution time of its actions, the duration of external processes, and by sensing features at appropriate intervals, not by consulting clocks.[2] We will provide more information about the execution semantics of CIRCA's real-time controllers below.

Given the above limitation on the form of the controller, the controller synthesis problem can be posed as *choosing a control action for each reachable state (feature-value assignment) of the system.* This problem is not as simple as it sounds, because the set of reachable states is not a given — by the choice of control actions, the CSM can render some states (un)reachable.

Indeed, since the CSM focuses on generating *safe* controllers, a critical issue is making failure states unreachable. In controller synthesis, this is done by the process we refer to as *preemption*. A transition t is preempted in a state s iff some other transition t' from s must occur before t could possibly occur. In the process of controller synthesis, the CSM achieves preemption by choosing a control action for the state that is fast enough that it is guaranteed to occur before the transition to be preempted.

Note that the question of whether a transition is preempted is not a question that can be answered based on local information: preemption of a transition, t in a state, s is a property of the controller as a whole, not of the individual state. For example, to know when a bomb is going to go off in a room with you, you can't just consider how fast you can throw the bomb out the window — you must also consider how long its timer has been running before you got to the state in which you will throw it out the window. It is this non-local aspect of the controller synthesis problem that has led us to use automatic verification.

Representing a control problem

CIRCA's State Space Planner 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 possible system states. These transitions are of four

[2] Of course, it would be possible to feature-encode certain key time periods for CIRCA's benefit by, for example, using a one-shot timer to set a register when that period has expired.

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.

For example, Figure 2 shows several transitions used in a situation where CIRCA is to control the Cassini spacecraft in Saturn Orbital Insertion.[3]

While in general there is no guarantee that an implicit representation like this will be smaller than enumerating the state space, in practice we find this representation far more efficient. The irredundant representation of processes (e.g., the process of IRU failure that can occur in any state in which the IRU is not already failed), is also easier to engineer. Furthermore, we use this implicit representation in concert with algorithms that allow us to avoid enumerating unreachable states, providing a further advantage.

CSM algorithm

At the highest level of abstraction, the controller synthesis algorithm is as follows:

1. Choose an element of the set of reachable states (at the start of controller synthesis, only the initial state(s) is(are) reachable).
2. Choose a control action (an action or a reliable temporal) for that state.
3. Invoke the verifier to confirm that the (partial) controller is safe.
4. If the controller is *not* safe, use information from the verifier to direct backtracking.
5. If the controller *is* safe, recompute the set of reachable states.
6. If there are no unplanned reachable states (reachable states for which a control action has not been chosen), terminate successfully.
7. If some unplanned reachable states remain, loop to step 1.

Figure 3 provides a simple "comic-book" illustration of the process of controller synthesis. Initially (i), there is only one state reachable, the initial state. In (ii), the CSM has chosen a control action (dashed line) for the initial state (planned states are shaded gray), that will carry the system to a goal state, *s1* (goal states are indicated by bold outlines). There is also a temporal transition (double line) that may carry the system to *s2*. In (iii), we see the CSM decide to assign *no-op* as the control action for *s1*. This is permissible because *s1* is a safe state (there are no transitions to failure from that state), and is desirable because *s1* is a goal state. In (iv), the CSM attempts to complete the controller synthesis process by assigning an action to *s2* that will carry the system to *s3*. However, this action does *not* preempt the transition to the failure state (black). This triggers a backtrack (v), and the system chooses an alternative action (vi) that will carry the system to *s1* (instead of *s3*). This alternative action *does* preempt the transition to the failure state (dark bar superimposed on the transition arrows), so the controller is safe. (vi) shows how the set of reachable states may vary as the controller synthesis process proceeds: at this point *s3* is no longer reachable, since the CSM has chosen not to employ the action that made it reachable in (iv). All reachable states have been planned for, so the controller synthesis process has terminated successfully.

During the course of the controller synthesis run above, the CSM will have employed the verifier module after each assignment of a control action (i.e., after

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

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

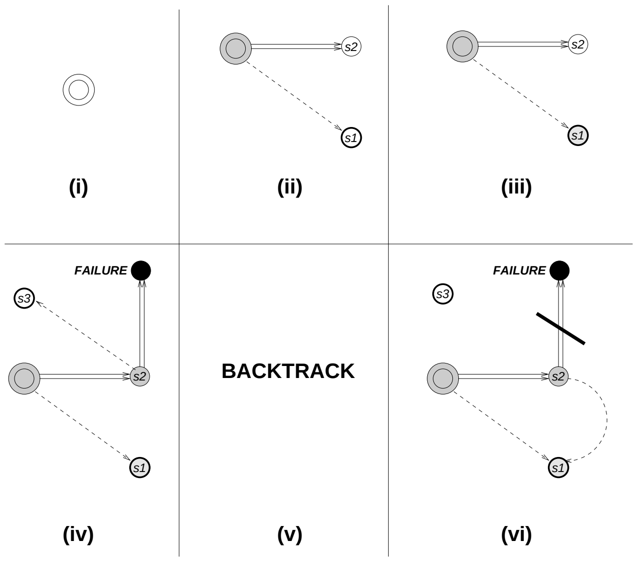

Figure 3: A simple example of controller synthesis.

ii, iii, iv and vi). However, at stages ii, iii and iv, the controller is not complete. At such points we use the verifier as a conservative heuristic by treating all unplanned states (e.g., *s2* in iii) as if they are "safe havens." Unplanned states are treated as absorbing states of the system, and any trace that enters these states ends and is regarded as successful. When the verifier indicates that a CSM-generated controller is *unsafe*, the CSM will query it for a path to the distinguished failure state. The set of states along that path provides a set of candidate decisions to revise.

Modeling for verification

In controller synthesis, the CSM uses a model which is oversimplified and is biased to be overoptimistic. It relies on the automatic verification system to assure that the controllers it builds are safe. This means it is critical that the verification system have a faithful model of the execution of the system and of the environment in which it operates. This section explains how we constructed this model.

Execution semantics

In order to model the of the RTS accurately, we must understand how its controllers are represented and executed. The controllers of the CIRCA RTS are not arbitrary pieces of software; they are intentionally very limited in their computational power.[4] The controller generated by the CSM is compiled into a set of *Test-Action Pairs* (TAPs) to be run by the RTS. Each TAP

[4]These limitations serve to make controller synthesis computationally more efficient and make it simpler to provide an operating platform that can provide timing guarantees.

```
#<TAP 8>
        Tests   : (AND
                    (TYPE_OF_CONVEYOR_PART SQUARE)
                    (PART_IN_GRIPPER NIL)
                    (EMERGENCY NIL))
        Acts    : pickup_known_part_from_conveyor
```

Figure 4: Sample Test-Action Pair from a CIRCA controller for a simulated PUMA robot arm attached to a conveyer belt.

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. A sample TAP for the Puma domain is given in Figure 4.

The set of TAPs that make up a controller are assembled into a loop and scheduled to meet all the TAP deadlines. The deadlines are computed from the delays of the transitions that the control actions must preempt.[5] It is possible that scheduling will not succeed. In this case, the AIS will backtrack to the CSM to revise the controller, generate and schedule a new set of TAPs.

Timed automata

Now that we have a sense of the execution semantics of CIRCA controllers, let us briefly review the modeling formalism, timed automata, before presenting the model itself. A timed automaton is a nondeterministic finite automaton (NFA) augmented with timing information. In the explication in this section, we follow Rajeev Alur's notation for describing timed automata (Alur 1998), and refer the interested reader to his paper for more details.

Definition 1 (Timed Automaton) *A timed automaton A is a tuple $\langle L, L^0, \Sigma, X, I, E\rangle$ where*

1. *L is a finite set of locations;*
2. *$L^0 \subseteq L$ is a subset of initial locations;*
3. *Σ is a finite set of labels;*
4. *X is a finite set of clocks;*
5. *I is a mapping, $L \longrightarrow \Phi(X)$ from locations to* clock constraints *(see below) and*
6. *$E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is the set of* switches *— transitions augmented with clock constraints ($\Phi(X)$), clock resets (2^X), and a label (Σ).*

The clock constraints that we use in our modeling will all be of the form $c_i \leq k$ or $c_i > k$ for some clock

[5]The tests and actions that the RTS can execute as part of its TAPs have associated worst-case execution times that are used to create and verify the TAP schedule.

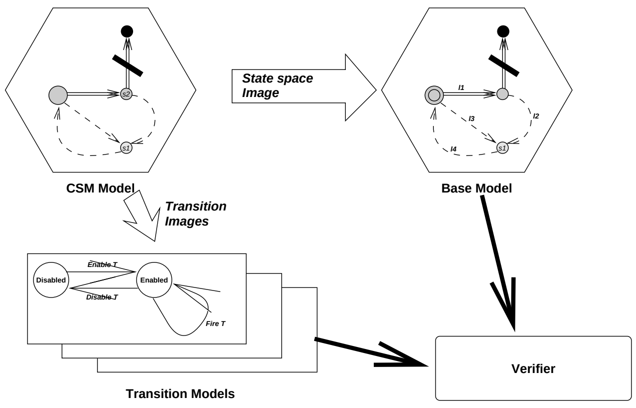

Figure 5: A pictorial summarization of the verifier model and its relation to the CSM model.

c_i and integer constant k. Note that while the clocks are always compared to integers, they may take on arbitrary real values; this is a *continuous time* model.

The labels (sometimes referred to as events) are used in the definition of products of timed automata, in order to synchronize switches in different machines. They are important to our modeling effort, since we explicitly model the multiple processes as separate automata.

Modeling CIRCA with timed automata

CIRCA translates the CSM model into a set of interacting timed automata for a timed automaton verifier (see Figure 5). There is one "base machine," the locations of which correspond to the states of the CSM model. The base machine captures the overall state of the system and its environment. The base machine interacts with a number of "transition machines," that correspond to the transitions the CSM reasons about. This interaction is captured by the labels on the transitions of the various machines; these ensure that the base machine state reflects the effect of the transitions and ensure that the state of the transition machines accurately indicate whether or not a given process is enabled in a particular system state. The use of multiple automata permits us to accurately and elegantly capture the interaction of multiple, simultaneously operating processes.

There are two classes of safety violations that we look to the verifier to detect. The obvious one is a transition to the CSM's distinguished failure state. The second is a failure to successfully preempt some transition that does not carry the system directly to a failure state. These are transitions that the CSM has decided to preempt in order to make other states unreachable, possibly to make the controller smaller and more efficient or to avoid other states from which the failure state will be reachable. To detect the second class of safety violations, for each state s, we add to the base machine a transition to the distinguished failure state for each transition that the CSM intends to preempt in s.

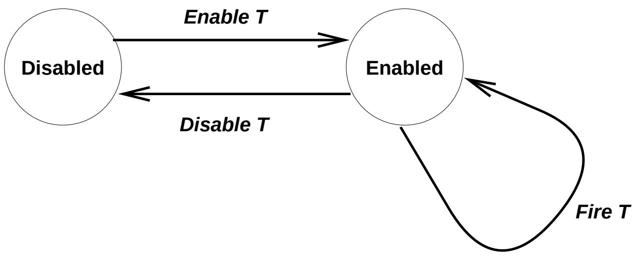

Figure 6: A timed automaton corresponding to a CSM transition, T.

The actual timing characteristics of the transitions are represented in separate transition machines. As we have outlined above, the CSM's automaton is a poor representation of the actual execution semantics of the CIRCA control system. There is not a single plant taking one action at a time; the environment is made up of a number of processes that are executing concurrently. These processes are represented as the nonvolitional transitions in a CSM domain description: the temporals, events and reliable temporals. The actions of the CIRCA control system occur concurrently with these environment processes.

Therefore, for each such process (CSM transition), we add a separate machine, with a distinguished clock, to the verifier model. These machines are all of the same form: they have two states, one for when the process is enabled, and one when it is disabled. These machines have three transitions: one from enabled to disabled, one in the opposite direction, disabling the process, and one self-arc from the enabled state corresponding to the completion of the process, when it has its effect on the environment.

The timing characteristics of the process are captured in the clock constraints of the transition machine. The *fire T* transition in the transition machine will contain a guard that expresses the lower bound constraint on the process. For an event or action, the guard will be vacuous. For a temporal, there will be a guard of the form $c_T \geq \Delta_{\text{min}}(T)$. For example, for `warm_up_IRU1` in Figure 2, there would be a constraint $c_{\texttt{warm\_up\_IRU1}} \geq 45\ (\Delta_{\text{min}}(\texttt{warm\_up\_IRU1}))$.

The guard on the transition only captures limits on how early a transition can occur. Upper bounds (on reliable temporals and actions) are captured by placing an invariant on the *enabled* state of the transition machine. For example, it is impossible to stay in the enabled state of the `warm_up_IRU1` machine for longer than 90 time units. The machine must leave this state, either by firing the transition, or by becoming disabled, before this time ($\Delta_{\max}(T)$) is up.

The interaction between the base machine and the transition machine for a CSM transition T is captured using labels. The arcs of the transition machine for T are given the labels *enable* T, *disable* T, and *fire* T (Figure 6). Those labels are added to appropriate transitions of the base machine as outlined below, in order to capture the enabling and disabling of processes, and the effect of those processes on the system and its environment.

The overall system state determines whether or not a given transition is enabled. For example, the transition `warm_up_IRU1` is only enabled in states when the `IRU1` feature has the value `warming` (Figure 2). Actions are enabled only in CSM states for which that action has been chosen for execution. This is modeled by tagging those transitions of the base machine that carry it from a state where T is enabled to one where it is disabled, with the *disable* T. This labeling will force the transition machine to its disabled state when that transition is followed. The enabling of processes is handled analogously, *mutatis mutandis*.

In the CSM model, there will be an edge from each state in which a transition T is enabled (and not preempted) to its successor state by that transition. For example, in states where the transition `warm_up_IRU1` is enabled, there will be arcs that carry the system to a new state in which the feature `IRU1` has changed its value from `warming` to `on` (Figure 2). In the construction of the base machine, the corresponding arcs will be labeled with the corresponding *fire* T, e.g., *fire* `warm_up_IRU1`. This ensures that the effects of transitions will be reflected in the state represented in the base model.

Final verification of complete controller

Recall that the descriptions of action transitions taken by controllers have associated with them delays that correspond to the worst case execution times of the action itself. For example, when controlling the Puma arm, we have worst-case execution times that correspond to the time it takes to actually move the arm. These execution times are not realizable in any actual system; they only serve as lower bounds on the actual times, which are functions not of the actions alone, but of the entire control program.

The actual worst-case time to execution of a TAP cannot be known until the entire controller is constructed because each TAP is a part of the overall reaction loop. This means that in general, the RTS will not be running the appropriate TAP immediately after the transition to the state for which its action should be run. In the worst case, the RTS might have to run (and fail) the tests for all other TAPs in the reaction loop before getting to the right one (see Figure 7).

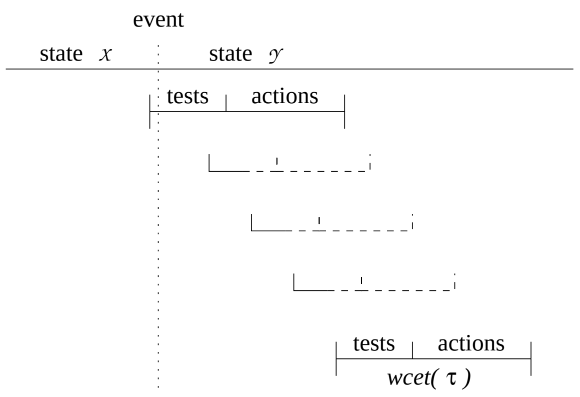

Figure 7: A worst-case execution of a particular TAP.

Figure 7 illustrates this possibility. The TAP that handles state Y has begun its tests just before the transition from X to Y. So the appropriate action will not be taken until all other TAPs (shown as half-solid bars, indicating that they do not run to completion) have failed their tests.

This shows, then, that the actual delay times for the various actions cannot be known until after the entire controller has been constructed *and compiled into a scheduled TAP loop*. Accordingly, the final step of controller synthesis is to re-run the verification process with a new model that contains, instead of the lower-bound estimates on action times, accurate worst-case bounds on the control actions, derived using the full reaction loop, with sensing actions as well as control actions.

Related Work

In independently-developed work, Asarin, Maler, Pneuli and Sifakis (AMPS) (1995; 1995) developed a game-theoretic method of synthesizing real-time controllers. They view the problem as trying to "force a win" against the environment, by guaranteeing that all traces of system execution will pass through (avoid) a set of desirable (undesirable) states. Their work stopped at the development of the algorithm and derivation of complexity bounds; it was never implemented. Our work concentrates on the implementation aspects of this problem and on making it computationally practical.

AI researchers at IRST have developed an approach to controller synthesis that they call "planning as model checking." (Giunchiglia & Traverso 1999) Their work is similar to our own in being concerned with *efficient* controller synthesis, but more like AMPS in that the verification system is used as a prover to solve the problem of controller synthesis, rather than being wrapped inside a synthesis program. Furthermore, the IRST researchers assume a simpler model of execution, ignoring the duration of actions, and the question of how the controller is to be implemented.

Kabanza, et. al. have developed work (Kabanza 1996; Kabanza, Barbeau, & St.-Denis 1997) very similar to ours in scope and intention. Their early work (fully presented in (Kabanza, Barbeau, & St.-Denis 1997)) is similar to the original CIRCA State Space Planner work, but does not take into account metric temporal information. Later work (Kabanza 1996), extends the original framework by incorporating metric time, but does so by effectively imposing a system-wide clock and progressing the controller one "tick" at a time. In control problems with widely varying time constants, this approach will lead to an explosion of states; we have adopted model-checking techniques that minimize this state explosion.

Conclusions

The CIRCA CSM is a novel application of automatic verification systems to automatic synthesis of controllers. Previous attempts to use automatic verifiers in controller synthesis have limited themselves to simpler execution semantics. Our system has a rich model of the execution of its timed controller, that reflects the behavior of a hard real-time executive.

The CIRCA system described above has been implemented as a Common Lisp program. The CSM currently works with two different verifiers, KRONOS (Yovine 1998) and a timed automaton verifier of our own that is tailored specifically to the timed automata CIRCA builds. KRONOS has better backward-verification and our own verifier has better forward verification (measured in terms of states explored). This means that KRONOS provides better performance at determining whether or not a given CIRCA controller is safe, but in the case where the controller is *not* safe our custom verifier is able to find a path to the failure state more rapidly. We are currently working both to improve our own verifier and to determine whether to use both verifiers together.

Other areas of current work include expanding the CIRCA framework to the synthesis of more complex controllers. We are currently looking at automatically synthesizing hybrid controllers where the continuous aspect of the problem is expanded to linear control of a set of continuous parameters, rather than simply clock constraints.

References

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

Asarin, E.; Maler, O.; and Pneuli, A. 1995. Symbolic controller synthesis for discrete and timed systems. In Antsaklis, P.; Kohn, W.; Nerode, A.; and Sastry, S., eds., *Proceedings of Hybrid Systems II.* Springer Verlag.

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.

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`.

Giunchiglia, F., and Traverso, P. 1999. Planning as model-checking. Paper accompanying invited talk to be presented at the 1999 European Conference on Planning (ECP-99). Available through `http://afrodite.itc.it:1024/~leaf/`.

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

Kabanza, F. 1996. On the synthesis of situation control rules under exogenous events. Appeared in the working notes of the 1996 AAAI Workshop on *Theories of Action, Planning, and Robot Control: Bridging the Gap.*

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.; Goldman, R. P.; Pelican, M. J.; and Krebsbach, K. D. 1999. SA-CIRCA: Self-adaptive software for hard real time environments. *IEEE Intelligent Systems* 14(4):23–29.

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.

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