Appears in Proc. IEEE Int'l Conf. on Robotics and Automation
San Francisco, CA April 22-28, 2000

Using Model Checking to Guarantee Safety in Automatically-Synthesized Real-Time Controllers

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

Introduction

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 [10, 11, 12], these controllers are synthesized *automatically* and dynamically, on-line, *while the platform is operating*. Unlike many other AI planning 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.

This paper is intended to convey an intuitive, qualitative understanding of the way CIRCA verifies its plans using model checking techniques. The mathematical details of the (existing) model checking techniques are described elsewhere [1, 13], and we are also writing a journal-length paper to provide a more explicit mathematical description of our approach.

Our work on model checking is unique in its focus on *synthesis*: model checking (and verification in general) have primarily been developed in the context of proving that a hand-designed system enforces some desired properties. For example, one classic verification problem involves proving that a gate controller for a railroad crossing will always lower the gate before a train crosses. In this example, the verifier is given a description of the train movement, a description of the gate movement, and a description of a hand-generated controller. The verification (or model checking) problem is to prove that certain states are never reached (e.g., that the train is never in the crossing area when the gate is up).

In contrast, CIRCA's State Space Planner (SSP) is responsible for actually designing the controller automatically. In the railroad gate example, CIRCA would be given descriptions of the train and gate dynamics, and a description of the failures that might result if the gate is not lowered while a train is crossing. In response, the SSP would search for a controller design that would make those failure conditions unreachable. We use model checking *within* this search process, as the "test" function in a heuristically-guided generate-and-test search. The SSP incrementally builds up a model of the reachable state space, as it reasons about uncontrollable transitions and decides what actions it should take in each state. Heuristics guide the selection of actions for states. After each choice of an action for a particular state, the SSP queries a verification algorithm to ensure that the currently-planned state space does not make failure reachable.

CIRCA can easily solve the railroad crossing example, but the state-space graphs are too large to present here. Instead, we will describe our model checking approach using an even simpler example drawn from the domain of controlling a UAV. In this example, our goal is to generate a controller that will successfully engage and disengage the UAV's evasive maneuver functions to defeat missile threats.

In the next section, we provide a brief description of the planning process and describe the temporal semantics of the automata. Then we discuss the problem of verifying the safety of a controller automaton, and present a brief overview of the model-checking

approach. We show how the SSP automaton is converted into a verification problem for the model checking mechanisms, and discuss optimizations used to reduce the complexity of model checking.

The CIRCA SSP

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

For example, Figure 1 contains the transition descriptions for a simple UAV control problem. The transitions describe a problem in which a UAV is attempting to follow a normal flight path (hence the `*goals*` statement). However, at any time during its flight, the UAV might be tracked by enemy radar. Some time after the initial tracking, a surface-to-air missile (SAM) may be launched. If no countermeasures are taken, that SAM may destroy the UAV after at least a certain minimum amount of time has passed (e.g., the minimum flight time of the missile). The UAV has available to it some evasive maneuvers that will cause the SAM to miss the UAV, if the UAV initiates its maneuvers quickly enough. Also, since the maneuvers divert the UAV from its nominal trajectory, the UAV should end its evasive behavior whenever possible.

Figure 2 shows the state space resulting from a simple timed controller design that will preserve the safety of the UAV. In the initial state, labeled "State 17" and shown as a shaded oval, the UAV is on its normal trajectory and has no indication of a radar-guided missile tracking it. This is a desirable state, so the controller will make no effort to leave it. However, at any time, a radar threat could occur, moving the system into

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

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

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

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

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

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

Figure 1: A simple domain description for a UAV threatened by radar-guided missiles.

state 16. The controller will react to this threat by taking evasive action, and maintaining the evasive maneuvers until the missile has been avoided (i.e., until the system has entered state 24). At this time the threat has been neutralized, and the system is free to return to its normal flight path. This controller was automatically generated by CIRCA, and the state diagram was generated from CIRCA data structures by the daVinci program [6].

There are several important aspects to note about this example state space model, or finite automaton. First, note that the automaton contains loops: the UAV may be threatened by more than one missile, and will remain in (or re-start) evasive maneuvers as long as it is threatened. Second, observe that time is not an explicit part of the state representation. This is critical to the compact representation of looping plans; if we included time in the state representation, then loops would not occur and persistent reactive control against an unpredictable or adversarial world would explode the state space. Instead, our automaton neatly encodes the continously-reactive behavior of the UAV in a compact, efficient, and automatically-generated form. Of course, the transitions do have temporal semantics, as described in Figure 1.

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

Temporal Transitions — Drawn as double arrows, temporal transitions represent uncertain processes that may lead to change, but only after at least some minimum amount of time has passed ($\Delta \geq min\Delta$). The only temporal transitions in our simple UAV example lead to failure, and are not shown in Figure 2 because the safety-preserving controller design makes failure unreachable.

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

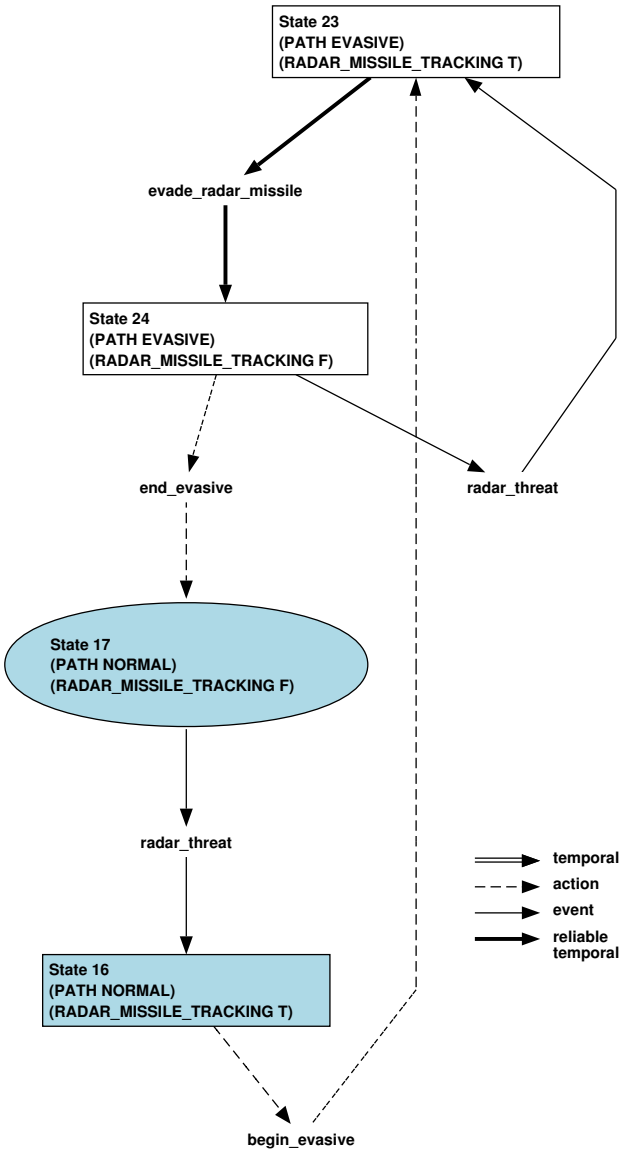

Figure 2: Simple UAV controller for evading radar-guided SAM threats.

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

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

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

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

Note that the `begin_evasive` action does not actually disable the `radar_threat_kills_you` transition: it simply begins the process of defeating the threat, which is represented by the reliable temporal transition `evade_radar_missile`. So if we were to draw the temporal transitions to failure (TTFs) in the graph of Figure 2, we'd see that the `radar_threat_kills_you` TTF is actually preempted out of both state 16 and the subsequent state 23. This is called a *dependent temporal chain*, because the amount of time left to preempt the TTF in state 23 is not the original minimum dwell time (as it was in state 16), but the original $min\Delta$ minus however much time the system may have dwelled in state 16 before transitioning to state 23. Since CIRCA reasons about worst-case circumstances, we can assume that that dwell time, in the worst case, is equivalent to the upper bound dwell time ($max\Delta$) imposed by the planned action `begin_evasive`, so that the new $min\Delta$ in state 23 is actually $1200 - 10 = 1190$.

Thus our temporal model is actually non-Markovian: the temporal semantics of the TTF out of state 23 depend on the path the system takes to get there. Naturally, this complicates the process of reasoning about the temporal model, and motivates our use of model checking to verify the required TTF-preemption properties.

Model Checking for Plan Verification

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

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

The problem with this algorithm is hidden in the definition of system state. To determine the possible successor states, we must know how long transitions have been enabled. For example, to determine at state 23 whether `radar_threat_kills_you` happens before or after `evade_radar_missile`, we must know whether the former transition has been active for 1200 time units before the latter has been active for 400 (see Figure 1).

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

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

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

By comparing this state against the problem definition given in Figure 1, you may readily see that this state is safe. `radar_threat_kills_you` cannot take place for 500 more time units, by which time `evade_radar_missile` will have preempted it.

Unfortunately, the verification problem, as naively framed, is not practically solvable. Since the clocks are integer-valued,[1] the set of system states is infinitely large. However, the set of interesting values is less than infinite, since there are only a finite number of decisions that need be made. For example, all values of `clock(radar_threat_kills_you)` that are over 1200 are equivalent. However, the number of relevant states may still be very large.

Timed Automata Representation Fortunately, researchers in computer-aided verification have found ways to compactly represent states like this for a class of finite state machines called *timed automata* [1]. Timed automata differ in a few minor ways from SSP state machines, but SSP state machines can be translated into timed automata. Timed automata states are composed of a *location* (corresponding to an SSP state, or feature vector) and a *clock-interpretation, or vector of clock values*. All of the clocks increment synchronously, but can be independently reset to zero by selected transitions. Transitions themselves are instantaneous. Temporal constraints in timed automata take two forms: transition *guard expressions* that must be true to enable a transition, and *state invariant* expressions that must be true all the time the system remains in a particular state.

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

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

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

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

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

Efficient Model Checking The critical concept for taming the complexity of timed automaton verification is an equivalence relation ("region equivalence") between system states [1]. This equivalence relation makes use of the intuition that all values for a given clock are equivalent above a maximum value (the largest constant the clock is ever compared to). Furthermore, since we are only concerned with the reachability of various states, the actual values of different clocks in a state are not as important as their *relative* values. Because the clocks are all notionally incremented at the same rate, the relationships between

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

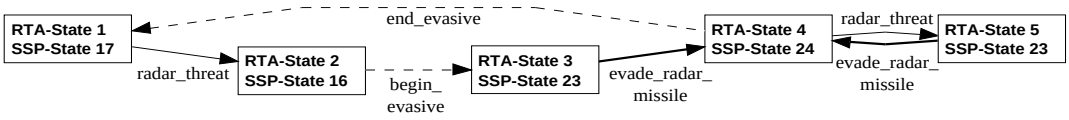

Figure 3: The RTA analysis for the example UAV domain.

the clock values upon entry to a state is sufficient to determine which outgoing transitions are possible: a clock that is behind another cannot catch up (within a state). Based on this equivalence relation, it can be shown that any timed automaton (SSP plan) has only a finite number of states.[2] Therefore, the problem of determining reachability (SSP plan verification) is decidable.

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

Therefore, we can compactly represent clock regions using a *difference-bound matrix* [5] whose entries represent bounds on the difference between pairs of clocks and between single clocks and a dummy clock whose value is always zero. Difference-bound matrices have two advantages. First, they provide a compact representation for equivalence classes of clock-states in timed automata. Second, they also have a canonical form, derived using any standard all-pairs shortest-path algorithm [5]. Putting the associated difference-bound matrices into canonical form makes it easy to determine when two automaton states are equivalent. Recognizing equivalent states is, in turn, necessary in order for reachability search to terminate.

Figure 3 compactly illustrates the reachability verification of the SSP plan given in Figure 1, optimized by the use of difference-bound matrices. A simple examination of Figure 3 shows one notable aspect of the RTA verification: there are two RTA states (3 and 5) that correspond to the SSP state 23. That is, the RTA algorithm has recognized the distinction between the two routes into SSP state 23 (see Figure 2) as being a temporally significant difference. The temporal transition to failure from state 23 will have different amounts of time left on its clock depending on whether we enter from state 16, where it was already enabled, or state 24, where it was not enabled. Thus the RTA algorithm is unrolling the important paths through dependent temporal chains, checking reachability of failure by removing the original non-Markovian temporal semantics.

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

Related Work

The CIRCA SSP is a reaction planner, and thus has much in common with work on reactive systems in AI and control theory. CIRCA is unusual in two ways: it automatically synthesizes, or plans, its reactions, and it provides performance guarantees through the methods of hard real time.

In independently-developed work, Asarin, Maler, Pneuli and Sifakis [2, 9] 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 method is very similar to ours, but 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.

Kabanza *et al.* have developed work [7, 8] very similar to ours in intention. Their early work (fully presented in [8]) is similar to the original CIRCA State Space Planner work, but does not take into account metric temporal information. Later work [7] 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.

Markov Decision Processes and Partially-Observable

Markov Decision Processes provide a theoretical basis for planning and action that is similar to discrete control theory, but they place the accent on uncertainty [4]. CIRCA simply represents uncertainty through nondeterminism: CIRCA transitions may have alternative outcomes; uncontrollable events may or may not occur; etc. The SSP techniques discussed in this paper do not attempt to reason about quantified measures of uncertainty, they make the worst-case assumption: "anything bad that can happen will happen." However, there has been some preliminary work on developing a probability model for CIRCA, to permit principled model-pruning decisions [3].

Conclusions

We have used several different model checking implementations within the SSP. The KRONOS model-checking tool [13] has proven very effective at identifying when failure is reachable, using a rapid backward-reasoning method that is quite different from the forward search described above. However, KRONOS is not particularly good at finding individual paths to failure, which are essential for guiding the SSP's backtracking. We have implemented a custom version of the difference-bound matrix methods described above, to rapidly identify paths to failure. Currently, we are designing a hybrid model checking approach that will use KRONOS as a rapid check, and invoke our algorithm to identify a culprit path if failure is reachable.

Our approach to synthesizing discrete controllers using model checking is innovative but costly. Even with the optimizations discussed above and others, invoking model checkers on large problems can still lead to very large state spaces. We are currently developing several new techniques to improve performance by taking advantage of the unique aspects of the CIRCA SSP problem. We are also using model checking to verify not just the SSP plan, but also how it is implemented in reactive execution rules by the CIRCA executive.

References

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

[2] E. Asarin, O. Maler, and A. Pneuli, "Symbolic controller synthesis for discrete and timed systems," in *Proceedings of Hybrid Systems II*, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, Springer Verlag, 1995.

[3] E. Atkins, E. H. Durfee, and K. G. Shin, "Plan Development Using Local Probabilistic Models," in *Uncertainty in Artificial Intelligence,Proceedings of the Twelfth Conference*, E. Horvitz and F. Jensen, editors, pp. 49–56. Morgan Kaufmann, August 1996.

[4] C. Boutilier, T. Dean, and S. Hanks, "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, July 1999.

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

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

[7] F. Kabanza, "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*, August 1996.

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

[9] O. Maler, A. Pneuli, and J. Sifakis, "On the synthesis of Discrete Controllers for Timed Systems," in *STACS 95: Theoretical Aspects of Computer Science*, E. W. Mayr and C. Puech, editors, pp. 229–242, Springer Verlag, 1995.

[10] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 6, pp. 1561–1574, 1993.

[11] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, March 1995.

[12] D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach, "SA-CIRCA: Self-adaptive software for hard real time environments," *IEEE Intelligent Systems*, vol. 14, no. 4, , July/August 1999.

[13] S. Yovine, "Model-checking timed automata," in *Embedded Systems*, G. Rozenberg and F. Vaandrager, editors, Springer Verlag, 1998.