

Imposing Real-Time Constraints on Self-Adaptive Controller Synthesis

David J. Musliner

Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive, Minneapolis, MN 55418*
david.musliner@honeywell.com

Abstract. Self-adaptive systems must reconfigure themselves, at runtime, to compensate for changing environments, objectives, and system capabilities. This paper discusses how the SA-CIRCA architecture for intelligent autonomous systems can automatically synthesize customized control software on the fly, and how that synthesis process itself can be managed to conform to real-time deadlines that may constrain the time available for reconfiguration. By restricting the scope of the problems it is trying to solve, by using incremental improvement algorithms, and by trading off solution quality against computation time, SA-CIRCA operates as a self-aware, self-adaptive system responding in real-time to perceived changes.

1 Introduction

Self-adaptive systems must reconfigure themselves, at runtime, to compensate for changing environments, objectives, and system capabilities. At first glance, this concept poses the major problem that the self-adaptive system must be able to understand its own objectives, capabilities, and environment in order to both detect changes and reconfigure itself. Upon further consideration, it becomes clear that this raw self-adaptation ability alone is not enough: the self-adaptive system must actually perform its self-adaptation online, as the deployed system is operating. Hence the real-time demands of the world are also applicable to the reconfiguration process itself! Clearly, it will be extremely challenging to deploy self-adaptive software into mission-critical applications, where any violation of real-time deadlines can have catastrophic consequences.

The Self-Adaptive Cooperative Intelligent Real-time Control Architecture (SA-CIRCA) can automatically synthesize customized controllers for autonomous systems, on the fly. This paper discusses how that synthesis process itself can be brought under soft real-time control, so that the synthesis of new controllers is accomplished within real-time deadlines (i.e., the time available for reconfiguration). The key to this control of deliberation is SA-CIRCA's ability to reason

* This work was supported by the Defense Advanced Research Projects Agency under contracts F30602-98-C-0212 and F30602-00-C-0017, and by a National Science Foundation Graduate Fellowship.

about resource restrictions and modify the control problems for which it synthesizes controllers. Because SA-CIRCA is able to explicitly and accurately reason about its own predictable performance, it can not only recognize overconstraining domains, it can also analyze the potential effects of various changes to its goals or plans. By restricting the scope of the problems it is trying to solve, by using incremental improvement algorithms, and by trading off solution quality against computation time, SA-CIRCA is designed to operate as a self-aware, self-adaptive system responding in real-time to perceived changes.

In the next section, we present a brief overview of the SA-CIRCA architecture and outline how its controller-synthesis algorithms work. We then discuss the various means available to control the complexity and resource usage of the controller synthesis algorithms, describing the performance tradeoffs that result. To demonstrate these effects, we present several implemented examples showing the tradeoffs that can be made to meet real-time restrictions.

2 Self-Adaptation via Automatic Synthesis of Controllers

Building on the proven CIRCA architecture for intelligent real-time systems [12, 13], SA-CIRCA is an architecture for intelligent self-adaptive systems that must meet real-time deadlines [14]. Many intelligent agent architectures (e.g., RAPs [4], PRS [7], 3T [3]) essentially provide customized programming environments that make it easier for humans to write complex programs that behave appropriately. In contrast, SA-CIRCA *automatically synthesizes* its control programs (or plans) from primitive descriptions of the system it is controlling, the system objectives, and the environment in which the system operates.

2.1 The Example Domain

To help the reader understand this distinction clearly, and to set the stage for later examples, consider the domain illustrated in Fig. 1. The Puma robot arm is assigned the task of packing parts arriving on the conveyor belt into the nearby box. The conveyor moves at a fixed rate and the parts are spaced apart on the belt so that they arrive with some maximum frequency. Once at the end of the belt, each part remains motionless until the next part arrives, at which time it will be pushed off the end of the belt (unless the robot picks it up first). If a part falls off the belt because the robot does not pick it up in time, the system is considered to have failed. Thus, the arriving parts impose hard deadlines on the robot's responses; it must always pick up parts before they fall off the conveyor.

The parts can have several shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not know a priori how to pack all of the possible types of parts. If parts of a new shape arrive, the system can stack those parts on the nearby table until it has derived an appropriate box-packing strategy. The derivation of the packing method may involve search algorithms with potentially unbounded complexity. This aspect of

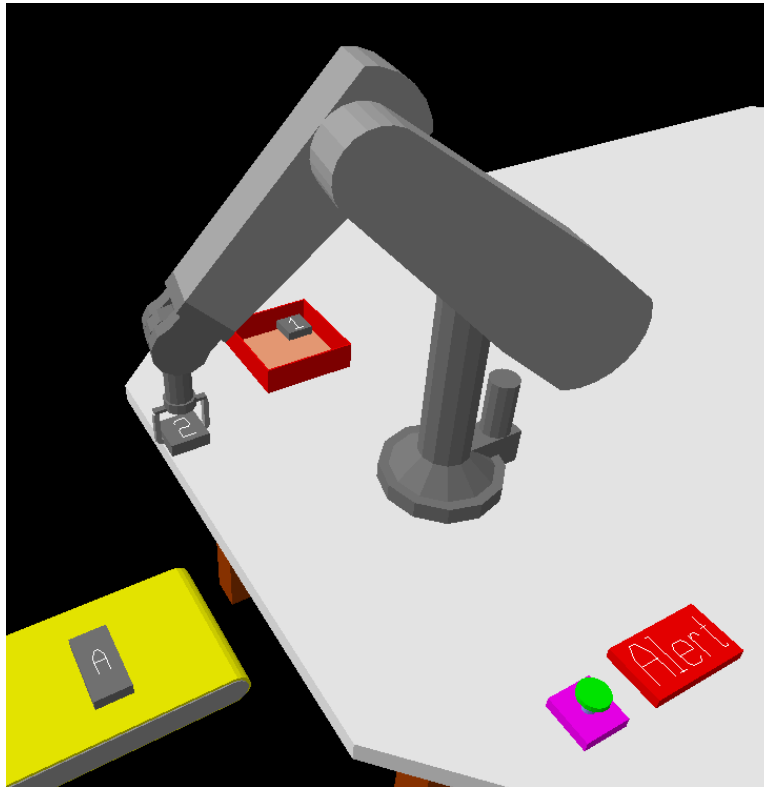


Fig. 1. The example Puma domain, in which the robot packs objects from the conveyor into the box.

the domain is used to exercise CIRCA's ability to combine arbitrary AI methods with real-time response guarantees.

The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system has only a limited time to push the button next to the light, or it fails. This portion of the domain represents a completely asynchronous interrupt with a hard deadline on its service time.

To cope with this domain properly, the robot control system must be able to provide real-time responses to unsynchronized domain events (part arrivals and emergency alerts) while also performing complex search algorithms (deriving packing methods and reaction plans in general). To complicate matters further, the speed of the Puma robot and the domain sensors are limited. Variations of the domain can be set up with different part arrival rates, emergency alert rates, robot speeds, etc. To be truly intelligent and real-time in this domain, the control system will need to be able to evaluate its capabilities, its goals, and the domain behavior restrictions. With that information, an intelligent system should provide some measure of useful performance, possibly involving trade-

offs that sacrifice aspects of the system behavior as necessitated by resource restrictions.

Applying a reactive control architecture such as PRS, RAPS, or 3T to this domain would require a human to write complex control rules specifying what the Puma should do in the many different situations that may arise. The human would have to build a control program (or plan, or set of concurrent behaviors) that manages the asynchrony of the environment. Even if the control program was written perfectly, however, none of these systems could provide guarantees that all the domain's real-time deadlines would be met. While PRS and the Rex/Gapps system [15, 8] can provide bounded response times on reactive execution, they have no way to reason about the timing characteristics of a domain to understand whether that bounded response time will be *fast enough*.

In contrast, applying SA-CIRCA to this domain requires the human to describe the robot capabilities, the environment processes, and the overall system goals. Rather than telling the system *when it should* pick up a part, the human simply tells the system that it *can* pick up a part, and that if it does not pick up a part quickly enough, it may fail. SA-CIRCA then automatically synthesizes a set of reactive rules and performs formal verification processes to guarantee that the domain's real-time deadlines are all satisfied by the new reactive controller.

2.2 Architecture Overview

Figure 2 presents a highly abstract view of the SA-CIRCA architecture, showing how the Adaptive Mission Planner (AMP), Controller Synthesis Module (CSM), and Real-Time Subsystem (RTS) interact to provide intelligent, adaptive, real-time control. All of the SA-CIRCA subsystems operate in parallel. At the top, the AMP reasons about long-term goals, problem structures, and approaching deadlines to decide what the near-term goals should be, and what problems the near-term reasoning should be focused on. For example, in the Puma domain the AMP is responsible for reasoning about what part-packing algorithms should be developed for different part types, and what new robot control plans are necessary to implement those part-packing algorithms.

The AMP sends subgoals and problem configurations to the CSM, which develops real-time controllers to accomplish near-term goals. The controllers are in the form of a set of reactive rules that specify what actions the system should take in all of the possible different world situations. Each reaction that is critical to avoiding a potential failure includes a timing constraint. For example, in the Puma domain, the control plan might say what to do if the emergency alert light goes on and the robot is not holding a block (*push the button!*), and how quickly that action must be taken (*before the emergency results in failure!*).

The CSM sends these control plans to the RTS, which reactively executes the automatically-generated plans and enforces guaranteed response times. Meanwhile, the AMP and CSM continue to reason ahead about future contingencies and phases of the problem. In the Puma domain, the AMP might try to predict what new types of parts will arrive, and generate suitable control plans in

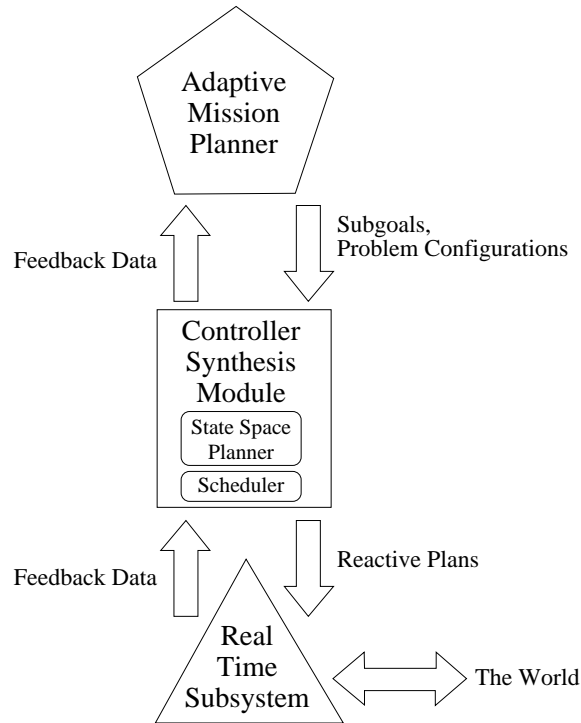


Fig. 2. The SA-CIRCA architecture includes three major components operating at different levels of abstraction and responsiveness.

advance. The RTS contains a plan cache mechanism that allows it to store additional reactive plans that are not currently executing, and switch to executing one of those plans with a (very fast) constant time operation.

Thus SA-CIRCA is designed to adapt its runtime behavior through both on-line, runtime planning and runtime selection of pre-built plans. In essence, if the AMP and CSM can build a plan ahead of time to handle a particular contingency, they will cache it in the RTS. If a contingency occurs for which no cached plan exists, the system will invoke its best available plan and the AMP will direct the CSM to synthesize a new, customized plan as quickly as possible. To the degree that a domain is accurately modeled and can be successfully controlled by the available execution resources, SA-CIRCA will provide safety guarantees and high-quality performance. When the real world diverges from the domain model, or the execution resources prove insufficient to control the domain, SA-CIRCA is designed to monitor its own performance, detect these problems, and gracefully degrade its plans to adapt.

The controller synthesis algorithms are extremely complex and potentially undecidable. Thus one of the keys to making CIRCA's safety guarantees is that each controller executed on the RTS must keep the system safe while waiting for

the CSM to generate the next controller. Some nicely-structured domains make it possible to build reactive controllers that preserve safety indefinitely while waiting for the CSM. For example, in some formulations of the Puma domain, the robot can respond to all types of arriving parts (by simply putting them down on the table) and all emergency alerts for an essentially unlimited amount of time. The only practical limitation is the number of parts the table can hold, and if this is made infinite, the domain allows perfect “holding patterns.” Of course, real worlds do not offer infinite tables (or infinite amounts of fuel for aircraft to circle in holding patterns).

Practical applications of the SA-CIRCA architecture will require the CSM to provide some level of predictable response, so that the RTS only needs to remain safe and stable for a limited time horizon. In this paper, our focus is on bringing the controller-synthesis activities of the CSM under this type of soft real-time control. When faced with limited time to reconfigure the RTS with a new reactive control plan, how can SA-CIRCA control its deliberation processes and ensure that it will synthesize a new controller in time?

3 Controlling the Synthesis Process

As illustrated in Fig. 2, the CSM includes a State Space Planner (SSP) and a Scheduler component. The SSP builds control plans based on a world model and a set of formally-defined safety conditions that must be satisfied by feasible plans [13, 6]. To describe a domain to SA-CIRCA, the user inputs a set of transition descriptions that implicitly define the set of reachable states. For example, Fig. 3 illustrates several transitions used in the Puma domain.

```

EVENT emergency-alert                ;; Emergency light goes on.
  PRECONDS: ((emergency F))
  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         ;; Pushing button cancels emerg.
  PRECONDS: ((part-in-gripper F))   ;; Requires empty gripper.
  POSTCONDS: ((emergency F))
  WORST-CASE-EXEC-TIME: 2.0 [seconds]

```

Fig. 3. Example transition descriptions given to SA-CIRCA.

The SSP builds plans by generating a nondeterministic finite automaton (NFA) from these transition descriptions. The SSP assigns an action to each

reachable state. These actions are selected to drive the system towards states that satisfy as many goal propositions as possible and to *preempt* transitions that lead to failure. For example, Fig. 4 shows graphically how a planned action can preempt a temporal transition to failure, if the action is constrained to *definitely occur* before the temporal transition *could possibly occur*. System safety is guaranteed by planning action transitions that preempt *all* transitions to failure [13]. Action assignments 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.

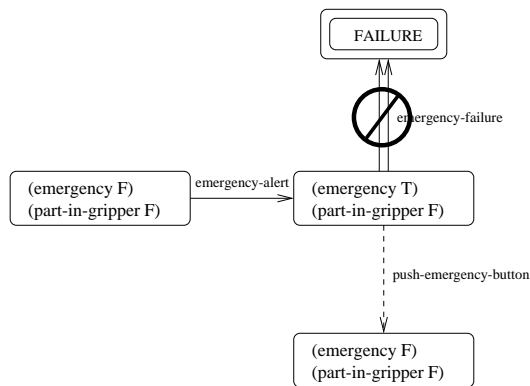


Fig. 4. Preemptive actions are planned to keep the system safe.

The control plan for the RTS is extracted from the set of planned actions in the NFA and cast in the form of Test Action Pairs (TAPs). Each TAP has a test expression that recognizes a subset of the NFA (world) states, and a single action that has been planned for those states. TAPs that preempt failures have timing constraints, and the Scheduler builds them into a fixed, looping schedule to be executed by the RTS. TAPs that do not preempt failures, but are planned only to achieve other non-safety-critical goals, are executed in a best-effort, “if-time” fashion.

3.1 Control Inputs

There are currently two different versions of the SA-CIRCA SSP: the evolved original SSP that reasons about nondeterministic worlds without explicitly representing quantitative uncertainty, and the Probabilistic State Space Planner (PSSP) from CIRCA-II [1, 2], that reasons explicitly about the probabilities of different transitions and states [10]. In both cases, the controller synthesis activity can be controlled by adjusting aspects of the SSP inputs: the problem configuration it is required to solve, and the parameters that describe precisely the nature of an acceptable solution.

For the non-probabilistic SSP, the problem configuration consists of :

Action Transitions — representing potential actions that the SSP can plan to invoke, with guaranteed results after some maximum amount of time.

Event Transitions — representing uncontrollable, instantaneous events.

Temporal Transitions — representing uncontrollable environmental processes that take at least some minimum amount of time.

Reliable Temporal Transitions — representing processes that are guaranteed to occur, given a certain amount of time.

Start States — describing possible states in which the system may start, or “wake up.”

Goals — describing desirable state features.

The Probabilistic State Space Planner (PSSP) uses additional input information, including probability rate functions associated with each transition, and a single probability threshold parameter that controls how conservative the PSSP should be in worrying about low-probability states.

3.2 Triggering Tradeoffs

SA-CIRCA has several ways of recognizing that the domain is overconstrained, and that the system cannot build a plan that accomplishes all of its goals and guarantees system safety. During the controller-synthesis process, the SSP may finish a complete search of the space of possible reaction plans and find that there are no suitable plans that can prevent failure. Or, if the SSP spends too much time trying to build a controller, it may time-out and be alerted by a timer interrupt. Finally, the SSP may come up with a set of desired reactions which are then rejected as unschedulable by the Scheduler. This is the most common way of recognizing an overconstrained domain: a suitable reaction plan exists, but its execution cannot be guaranteed with the limited resources of the RTS. At this point, the SSP would backtrack by default to make a different choice and produce a modified reaction plan. Alternatively, the AMP might decide to make a tradeoff instead, simplifying some aspects of the control problem so that the CSM can generate a feasible controller.

We have designed several experiments to illustrate and evaluate this tradeoff capability. Note that these tradeoff methods are not heuristics themselves; they can be implemented by simple procedures making bounded changes to the SA-CIRCA data structures describing the world model, the current control plan, etc. Furthermore, the effects of those changes are well-understood; SA-CIRCA can explicitly reason about the impact of applying its various tradeoff methods on the system’s performance. However, choosing *which* tradeoff method to apply in a particular situation remains a heuristic decision we have not yet addressed.

3.3 Ignoring a Temporal Transition to Failure

One of the most obvious and powerful tradeoffs is to simply delete or ignore one or more temporal transitions that lead to failure in the world model. This

corresponds to the SSP not even considering that some ongoing process will ever lead to failure. As a result, the TAP that was planned to preempt that temporal transition to failure (TTF) may be affected in several ways. In the following material, we will examine in detail one example of the types of performance tradeoffs that result from simply ignoring a TTF. We then outline several other possible outcomes, but do not investigate them in depth because they represent minor variations.

One Result of Ignoring a TTF If the SSP is not told about a TTF from a particular state, it is possible that the SSP will still choose the same action for that state, but that the action will no longer be preventing a failure. Because the action is not preempting a TTF, it will be implemented by an if-time TAP (see Sect. 3) that does not need to be scheduled, so the scheduling problem will be easier. However, performance will suffer because the system no longer guarantees to execute the affected TAP and prevent one particular type of failure.

In the Puma domain, for example, the AMP might decide to ignore the possibility of a part falling off the conveyor, perhaps because it is highly unlikely that the part will really fall. As a result, when examining a state in which a part is waiting on the conveyor, the SSP will no longer be required to plan a **pickup-part-from-conveyor** action to avoid failure. However, the action will still be planned because it is useful in achieving the system’s goals: the robot must pick up the part in order to pack it in the box, which satisfies the goal (**part-in-box T**). Note that the system can still make guarantees about other types of failures. For example, it can guarantee that it will avoid failures resulting from the emergency alert, because the actions preempting the **emergency-failure** TTF are still guaranteed.

Schedulability Effects of Ignoring a TTF To quantify the effects of this type of tradeoff method, we make the CSM try to build controllers for parametric variations of the Puma domain. Figure 5 shows the effect on schedulability over a range of arrival rates for emergency alerts and parts. If the arrival rates match a point below the lower, “normal plan” curve, then the system can build a schedule that will guarantee to both avoid emergency failures and prevent parts from falling off the conveyor. The form of this curve illustrates the tradeoff that the scheduling mechanism can make between tasks; when the emergency rate is relatively high, the system will still build a schedule, as long as the part arrival rate is sufficiently low that the Scheduler can allocate more resources to the tasks that respond to the alert. Conversely, when the emergency rate is lower, the system can deal with a faster rate of arriving parts. If the arrival rates match a point above the lower curve, then the system cannot build a schedule that will guarantee to avoid both emergency failures and dropping parts. However, if the system ignores the **part-falls-off-conveyor** TTF, then it can build guaranteed schedules for all of the instances below the upper line, the maximum rate of emergency alert arrivals that can be handled with the given primitives. The part arrival rate is no longer critical to the scheduling problem,

because the **pickup-part-from-conveyor** TAP, with a period determined by the **part-falls-off-conveyor** TTF, is no longer being scheduled and guaranteed.

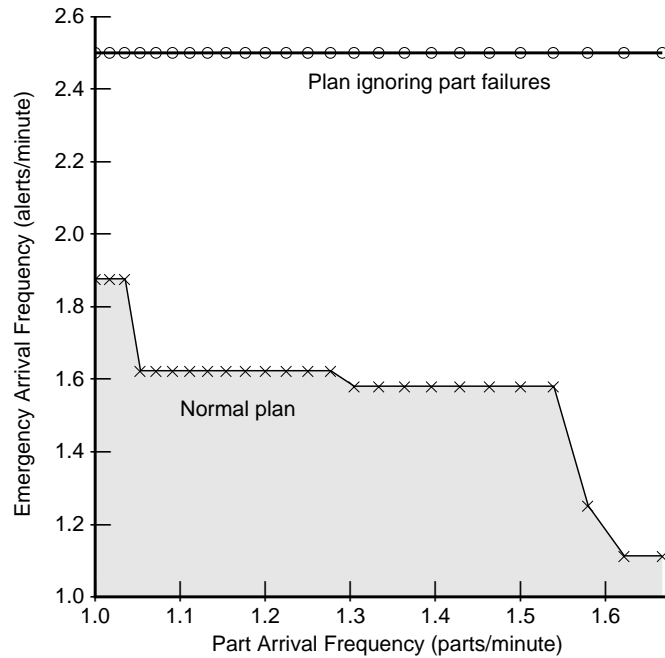


Fig. 5. The improved schedulability achieved by ignoring a TTF.

Performance Effects of Ignoring a TTF To illustrate the non-guaranteed nature of the resulting behavior, we implemented this tradeoff method in the Puma domain, increasing the rate of emergency alerts and part arrivals so that the original plan of actions is not schedulable. The SSP was told that the worst-case rate was 2.4 alerts/minute, well above the limit of schedulability for the entire problem, as shown in Fig. 5. The AMP then removes the **part-falls-off-conveyor** TTF from the SSP’s world model, re-plans, and builds a new TAP plan in which the **pickup-part-from-conveyor** action is implemented by an if-time TAP rather than a guaranteed TAP. Figure 6 illustrates the results from several hundred trials on the Puma simulator, with emergency alerts arriving with random delays uniformly distributed in the range of 25 to 30 seconds (2 to 2.4 alerts/minute). As we expected, when parts arrived more frequently, the number of parts falling off the conveyor would increase, as the system had less and less free time to apply to if-time behaviors.

Interestingly, the if-time TAP that implemented the **pickup-part-from-conveyor** action was always executed frequently enough to prevent failures when the parts arrived no more frequently than two parts/minute. Referring back to Fig. 5, we can see that the actual simulated execution was more robust than the Scheduler predicted; the “normal plan” plot in the graph indicates that even with the **pickup-part-from-conveyor** TAP in the schedule, the system could not be guaranteed to handle part arrivals faster than about 1.65 parts/minute. This result may indicate that the worst-case situations the SSP considered never occurred in the tests, or that the Scheduling or SSP algorithms were overly conservative. We plan to investigate further.

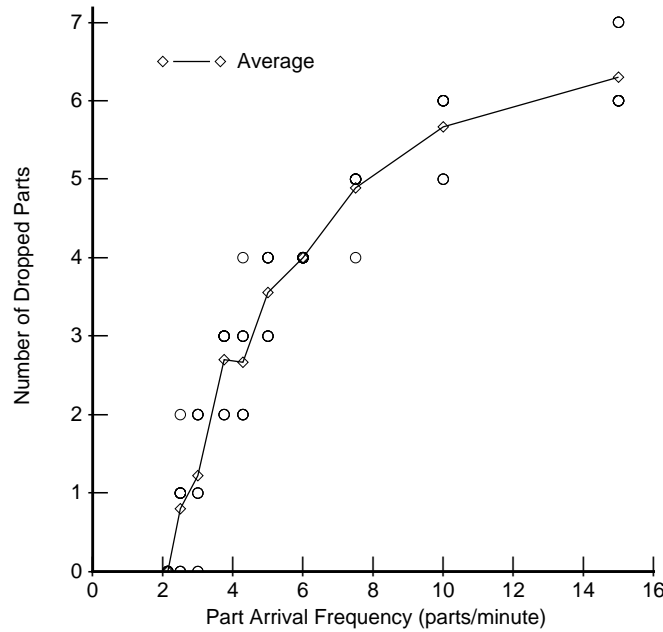


Fig. 6. The system behavior after ignoring the **part-falls-off-conveyor** TTF, with the emergency light frequency between 2 and 2.4 alerts/minute.

Alternative Results of Ignoring a TTF Several other outcomes are possible when the AMP chooses to ignore a TTF. For example, it is possible that, by ignoring one TTF from a state, a different temporal transition becomes dominant and still causes the planned action for that state to meet a deadline. In general this will mean that the deadline for the planned reaction will be longer, but the

TAP will still need to be scheduled. The resulting tradeoffs are similar to those above, in that the system can no longer guarantee to avoid all types of failures. In this case, however, no TAPs are moved out of the guaranteed list: instead, the reaction deadline of one of the TAPs will be increased, thus decreasing the desired utilization, and making the scheduling problem easier.

In the extreme case, ignoring a TTF may cause the planner to completely eliminate one or more planned actions, thus removing TAPs from the list to be scheduled. If an action was only planned originally to preempt failure (or as a “precursor” to that preemption), and was not instrumental in achieving any other system goals, then the action may be removed entirely. For example, if SA-CIRCA ignores the **emergency-failure** transition in the Puma domain, it will completely alter the world model and avoid planning the **push-emergency-button** action. Depending on the frequency of part arrivals, it may also eliminate the need to put parts on the table temporarily, and thus ignoring this one TTF could also remove the **stop-moving** and **place-part-on-table** actions from the plan. These latter actions are precursors that were included in the plan to establish the preconditions of the action that was planned to preempt the **emergency-failure** TTF, and thus they are also unnecessary.

While moving a TAP to the if-time list means that, in non-worst-case situations it may be still executed quickly enough, deleting a TAP altogether provides no such potential. Since if-time TAPs do not use any resources when the RTS is pressed for time, avoiding building if-time TAPs does not save any significant RTS execution-time resources. However, this tradeoff can significantly reduce the complexity of the controller synthesis problem, and thus it provides a powerful method for managing the CSM’s responsiveness.

An important feature of this tradeoff method, and of the SA-CIRCA approach in general, is that the system can introspectively examine the predicted effects of a particular tradeoff. In other words, SA-CIRCA might evaluate the worth of various tradeoff methods by examining the expected results in the world model. If the AMP considers ignoring a TTF, it can immediately recognize that the failure resulting from that TTF will be possible with the modified TAP plan. In addition, the AMP can examine the new world model and TAP plan to recognize more detailed aspects of the tradeoff. For example, if the new plan still includes all the same guaranteed TAPs as the original plan, then the AMP can conclude that the reaction previously planned to preempt the TTF is still being enforced, but at a lower rate. If the AMP knows that the worst-case rate of the ignored TTF is rarely achieved, this tradeoff option may be very attractive, because it has exchanged a decrease in one TAP’s response rate for the ability to schedule and guarantee the entire reactive plan.

3.4 Ignoring an Event Transition

Just as the AMP may decide to alter its treatment of temporal transitions, it may also choose to change how it considers event transitions. Ignoring an event transition may have many of the effects described above for temporal transitions: it may cut off parts of the world model state space, possibly making some goals

unreachable. Ignoring an event transition can thus reduce the planning time and decrease the number of TAPs planned, allowing the system to make guarantees for some subset of desired behaviors which were not previously schedulable.

For example, in the Puma domain, ignoring the **emergency-alert** event transition provides a large reduction in the planning time, because many states are eliminated from the model—in fact, the state space for our running example is reduced from 330 enumerated states and 158 reachable states to 106 enumerated and a mere 58 reachable states. Furthermore, a large number of contingency reactions are eliminated from the plan, and thus the complexity of the TAPs is reduced, and the scheduling problem is eased. Because the emergency alert is no longer of concern, the system is able to react to parts on the conveyor belt even more quickly than if the predicted alert rate is very slow (as in the extreme right edge of Fig. 5). While the example of Fig. 5 could handle parts arriving at most every 36 seconds, the plan built by ignoring the **emergency-alert** transition can handle parts arriving every 27 seconds, a significant improvement in capacity. Of course, the tradeoff is that the system is no longer monitoring the emergency light, and it will not react to an alert. If the AMP thinks that an alert is unlikely, or finds that the cost of failing to respond to an alert is sufficiently low, it may judge that the reduced planning time and improved part-packing reaction time are worth the risk involved in ignoring alerts.

More generally, we can see that ignoring an event transition can have the desirable effects of reducing the SSP’s planning time and simplifying the scheduling problem. The disadvantage, of course, is that this tradeoff method removes planned contingency actions entirely, as opposed to just moving the relevant TAPs to the if-time list (as ignoring a TTF can do). Because event transitions represent instantaneous events in the world, as opposed to the ongoing processes represented by temporal transitions, it seems plausible that the AMP could have knowledge of event probabilities that would be helpful in guiding the use of this tradeoff method. Ignoring highly improbable event transitions would obviously be a good approach, in order to ensure that the system is least likely to encounter world situations for which it is not prepared. The PSSP takes this idea one step further, using probabilistic information to provide uniform, unified pruning of the SSP model when necessary.

3.5 The Probabilistic State Space Planner

At the University of Michigan, colleagues have been working on a new version of the SSP that uses probabilistic information to guide the system in considering the most-probable states first [10]. The Probabilistic State Space Planner (PSSP) builds partial plans that are only probabilistically safe, because they only consider those states whose likelihood is above a given threshold. By explicitly pruning least-probable areas of the state space, the PSSP approach allows SA-CIRCA to optimize its allocation of controller-synthesis effort and runtime reactive resources against the most-probable types of failure. The AMP can adjust the probability threshold to alter the complexity of the controller synthesis problem. Moreover, if additional planning-time resources are available and the

RTS capabilities are actually the limiting factor, the system can build contingency plans to handle pruned areas of the state space and swap those plans in when the pruned, less-likely situations arise.

The PSSP techniques resemble work on developing policies for Markov Decision Processes, with the added complexity of a non-Markovian temporal model and the resulting looping automata. This problem formulation leads to more compact world models, but complex reasoning.

The probabilistic approach is less coarse and “heavy-handed” than simply ignoring an entire TTF. For example, the PSSP may realize that in one world state a TTF is highly unlikely (and thus can be ignored), but that same TTF is more likely in a different state (and should be handled). Furthermore, this approach retains the advantages of the general CIRCA approach, in that the system can introspectively examine the predicted effects of a particular tradeoff. We have not yet implemented any experiments in the Puma domain to demonstrate and evaluate this approach. However, tests in related domains (e.g., controlling simplified autonomous aircraft) show promise for this approach.

3.6 Modifying Action Transitions (Method Selection)

The AMP can also make changes to the action transitions that are available to the CSM for use in synthesizing controllers. The AMP may have several different methods for performing an action (or a test), and it can choose amongst them according to the resources available. For example, suppose that the Puma control system provides the RTS with two different implementations of the **place-part-in-box** operator: a slow, high-accuracy, “fine-motion” version; and a faster, lower-accuracy, “coarse-motion” version. Using the fine-motion operator allows the system to place the parts very close together, thus yielding densely-packed boxes. But the fine-motion operator needs four seconds to finish the placement operation. Using the coarse-motion operator requires the system to leave more space between the parts, since the placement is less-certain. As a result, the system will produce less-densely packed boxes, but it can produce them more quickly, because the coarse-motion operator only needs 2.5 seconds. Thus, in this example, selecting different operators will allow the system to trade off the quality of its results (the packing density) for the timeliness of its long-term and short-term behaviors (the speed of packing whole boxes and individual parts). This tradeoff method is equivalent in many ways to the “configuration selection” [9], “version selection” [11], and “design-to-time” [5] approaches. Given the faster coarse-motion operator, the system may be able to guarantee to respond in time to a higher frequency of emergency alerts than with the slower operator.

Experimental Results of Method Selection To provide a more quantitative demonstration of this tradeoff, we ran experiments using the coarse/fine operators described above. The fine-motion operator was defined to require no space at all surrounding parts being placed in the box: essentially, it could achieve 100% packing density with a fortuitous series of part arrivals. The coarse-motion operator, on the other hand, required one inch of clearance on all sides of the parts

in order to place them in the box. Naturally, the achievable packing density is lower with this operator, since parts necessarily occupy spaces larger than their actual size.

Figure 7 shows the improvement in response-time achieved by using the coarse-motion operator, displayed here by the increased rate of emergency alerts and part arrivals that can be handled. The lower curve shows the performance for the fine-motion operator used in the earlier examples (and previously graphed in Fig. 5). The upper curve shows the larger range of domains that can be handled using the faster, coarse-motion packing operator. The coarse-motion operator reduces the time allocated to the **place-part-in-box** TAP, and therefore the system can respond in time to more frequent part arrivals, emergency alerts, or both.

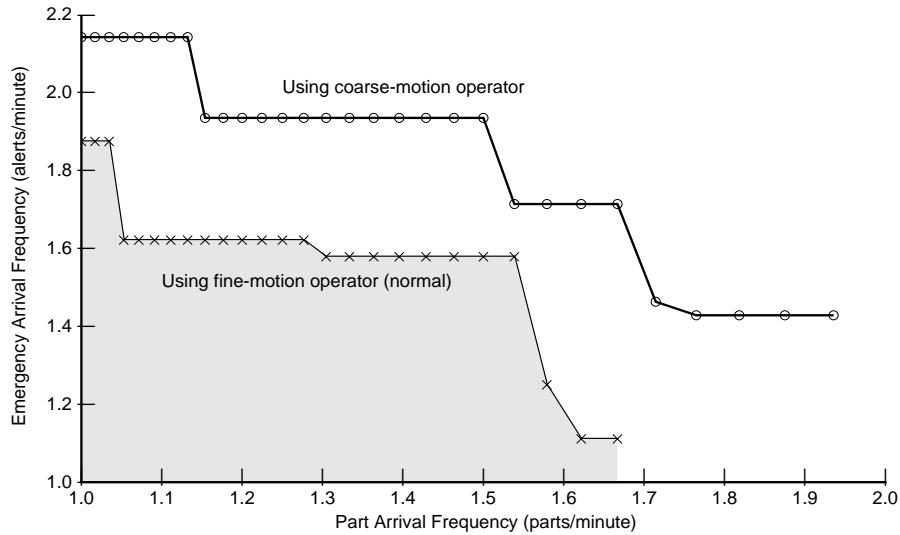


Fig. 7. Schedulability variations using different TAP implementations.

However, Fig. 8 shows the corresponding decrease in performance quality that resulted from the coarse-motion operator, when applied to 100 trials using randomly ordered arrivals of four different part shapes. On average, the density of the packed box was reduced from 70% using the fine-motion operator to 59% with the coarse-motion operator. In these experiments, simulations of the box-packing algorithm were continued until the first arrival of a part that did not fit in the box. The fine-motion version was able to pack an average of 45 parts in the box, while the coarse-motion version packed an average of only 26 parts.

Thus we can see that the improved schedulability and response time illustrated in Fig. 7 are only achieved at the cost of stiff performance degradation.

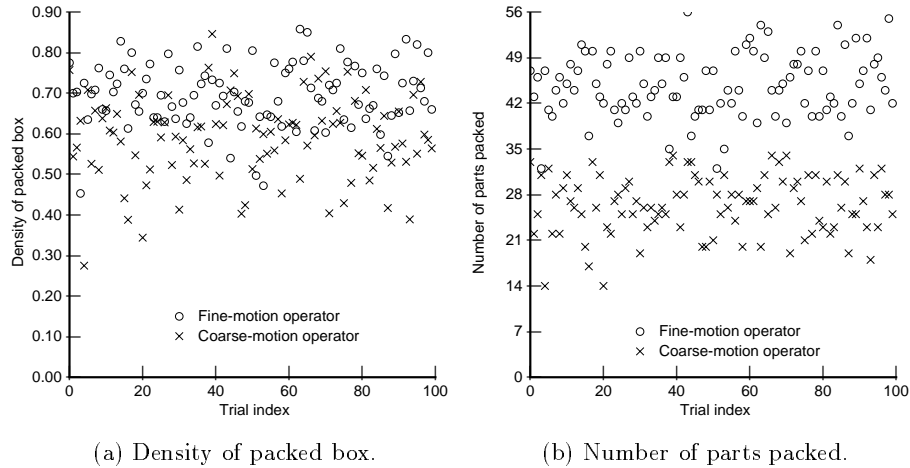


Fig. 8. Performance variations using different TAP implementations.

Generalizing Method Selection To use the method selection approach, the system obviously must have alternative methods for implementing feature tests and actions on the RTS. In addition, to make intelligent decisions about method selection, the system would require performance information describing the output quality and resource requirements of each method. This information could be relatively simple, or could be as complex as a full performance profile. In any case, because method selection retains the consideration of all world model states and does not remove any TAPs from the schedule, it is one of the more subtle tradeoff techniques, capable of altering the resource needs of the system without drastic effects on its performance guarantees. Depending on the assortment of different methods available, the method-selection approach can alter almost any quality measure of the reactive system's performance, including precision, accuracy, etc.

4 Summary and Future Directions

The ability to make performance tradeoffs in the face of resource limitations is a fundamental requirement for intelligent real-time systems, since the very nature of real-time domains includes resource constraints. We have presented a variety

of ways in which SA-CIRCA can make such performance tradeoffs, actively managing its allocation of deliberation and reaction resources. Our experiments in the Puma domain have demonstrated several different strategic tradeoff methods that the AMP can use to modify the problems it poses to the SSP, adjusting the difficulty of the controller synthesis problem, and thus the performance of the SSP itself.

In the probabilistic version of the SSP, the fundamental technique for controlling the search process is adjusting the threshold of state probabilities below which the SSP ignores states. An alternative or supplemental approach, which has not yet been implemented, is to impose a time horizon limit on the SSP's projective search as well. An explicit time horizon would tell the SSP that it only needs to build a reactive controller that can keep the system safe for a limited amount of future time, and this could be used to truncate the SSP search forward through the space of possible future worlds. In fact, if the AMP could tell for sure what would be an appropriate and feasible time horizon for a particular SSP problem configuration, these horizon limits would be a feasible simplification, not reducing the completeness or performance guarantees of an SSP's plan. As such, time horizon control could be even preferable to probability threshold adjustments for domains in which the expected progress through a mission plan can be sufficiently predicted.

References

- [1] E. Atkins, R. H. Miller, T. Van Pelt, K. D. Shaw, W. B. Ribbens, P. D. Washabaugh, and D. S. Bernstein, "Solus: An Autonomous Aircraft for Flight Control and Trajectory Planning Research," in *Proc. American Control Conference*, volume 2, pp. 689–693, June 1998.
- [2] E. M. Atkins, *Plan Generation and Hard Real-Time Execution with Application to Safe, Autonomous Flight*, PhD thesis, University of Michigan, 1999.
- [3] R. P. Bonasso, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents," in *Journal of Experimental and Theoretical AI*, 1996.
- [4] R. J. Firby, "An Investigation into Reactive Planning in Complex Domains," in *Proc. National Conf. on Artificial Intelligence*, pp. 202–206, 1987.
- [5] A. Garvey and V. Lesser, "Design-to-time Real-Time Scheduling," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 23, no. 6, , 1993.
- [6] R. P. Goldman, D. J. Musliner, K. D. Krebsbach, and M. S. Boddy, "Dynamic Abstraction Planning," in *Proc. National Conf. on Artificial Intelligence*, pp. 680–686, 1997.
- [7] F. F. Ingrand, M. P. Georgeff, and A. S. Rao, "An Architecture for Real-Time Reasoning and System Control," *IEEE Expert*, pp. 34–44, December 1992.
- [8] L. P. Kaelbling and S. J. Rosenschein, "Action and Planning in Embedded Agents," in *Robotics and Autonomous Systems 6*, pp. 35–48, 1990.
- [9] T.-W. Kuo and A. K. Mok, "Load Adjustment in Adaptive Real-Time Systems," in *Proc. Real-Time Systems Symposium*, pp. 160–170, December 1991.
- [10] H. Li, E. Atkins, E. Durfee, and K. Shin, "Resource Allocation for a Limited Real-Time Agent Using a Temporal Probabilistic World Model," in *Working Notes of the 2000 AAAI Spring Symposium on Real-Time Autonomous Systems*, 2000.

- [11] N. Malcolm and W. Zhao, "Version Selection Schemes for Hard Real-Time Communications," in *Proc. Real-Time Systems Symposium*, pp. 12–21, December 1991.
- [12] D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1561–1574, 1993.
- [13] 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.
- [14] D. J. Musliner, R. P. Goldman, M. J. Pelican, and K. D. Krebsbach, "Self-Adaptive Software for Hard Real-Time Environments," *IEEE Intelligent Systems*, vol. 14, no. 4, pp. 23–29, July/August 1999.
- [15] S. J. Rosenschein and L. P. Kaelbling, "The Synthesis of Digital Machines with Provable Epistemic Properties," in *Proc. Conf. Theoretical Aspects of Reasoning About Knowledge*, pp. 83–98, 1986.