To appear in Proc. Symposium on AI in Real-Time Control
Valencia, Spain, October 1994

AUTOMATING THE DESIGN OF REAL-TIME REACTIVE SYSTEMS

DAVID J. MUSLINER*, **KANG G. SHIN**** and **EDMUND H. DURFEE****

* *Institute for Advanced Computer Studies, The University of Maryland, College Park, Maryland 20742, USA. Email: musliner@umiacs.umd.edu.*

** *Department of EE & Computer Science, The University of Michigan, Ann Arbor, Michigan 48109, USA. Email: {kgshin,durfee}@eecs.umich.edu.*

Abstract. The Cooperative Intelligent Real-time Control Architecture (CIRCA) automates the process of designing, scheduling, and executing real-time reactive monitoring and control systems. This paper provides an overview of CIRCA from a design-automation perspective, and illustrates the architecture's ability to dynamically alter its control system design based on resource limitations or environmental constraints.

Keywords. Artificial intelligence; real-time computer systems; robots; control system design; self-adapting systems.

1. INTRODUCTION

The Cooperative Intelligent Real-time Control Architecture (CIRCA) (Musliner *et al.* 1993, Musliner *et al.* 1995) is designed to automate the entire process of building a real-time reactive monitoring and control system, from planning tasks, to deriving their constraints, to scheduling them, and finally to executing them predictably. By automating this design and implementation process, CIRCA is "intelligent *about* real-time." That is, CIRCA uses AI methods to dynamically and flexibly develop and modify its real-time behavior in the face of changing goals, capabilities, and/or domains. While many real-time AI systems can only promise "best-effort" performance, CIRCA is able to make explicit guarantees about its ability to achieve its goals within particular domains using limited sensor, processor, and actuator resources.

As illustrated in Fig. 1, CIRCA consists of several parallel subsystems. The AI Subsystem (AIS) is responsible for using complex AI methods to reason about a world model, deriving appropriate monitoring and control reactions for the system. These reactions are built into an execution schedule by the Scheduler module, and then downloaded to the Real-Time Subsystem (RTS). The RTS is designed to provide a predictable execution environment which can enforce hard real-time response guarantees for the planned reactions.

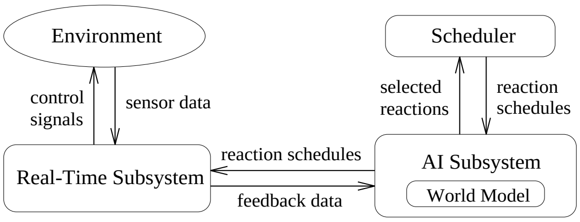


Fig. 1. CIRCA, the Cooperative Intelligent Real-Time Control Architecture.

The development of CIRCA makes several contributions to the state of the art, including a world model and planning algorithm tailored to the needs of hard real-time environments (Musliner *et al.* 1995), and a structured interface through which the arbitrarily complex AI planning subsystem can communicate with and control the predictable, guaranteed RTS (Musliner *et al.* 1993). This paper focuses on describing the overall operations of CIRCA from an automated-design perspective, emphasizing the way CIRCA retains predictable real-time behavior while also providing the flexibility and adaptability required for intelligent real-time control.

The work reported in this paper was supported in part by the National Science Foundation under Grants IRI–9209031 and IRI–9158473, by a NSF Graduate Fellowship, by the Office of Naval Research under Grant N00014–91–J–1115, and by the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039). David Musliner is also affiliated with the University of Maryland Institute for Systems Research (NSF Grant CDR-88003012).

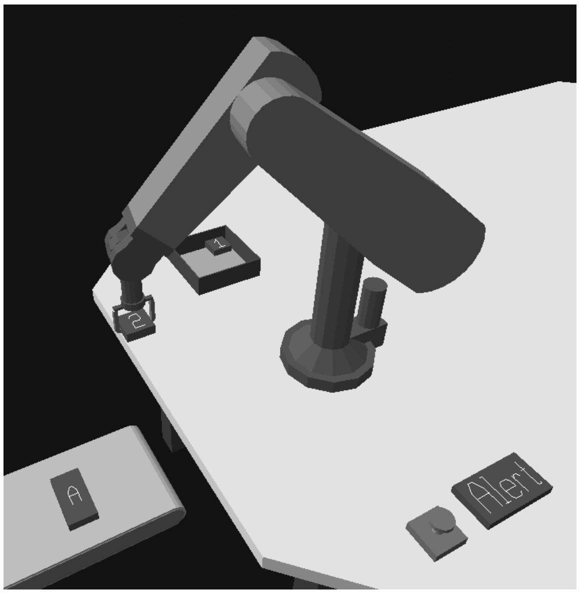


Fig. 2. The example Puma domain.

Examples of CIRCA's adaptive design behavior will be drawn from the prototype implementation controlling a simulated Puma robot arm which must pack parts arriving on a conveyor belt into a nearby box (see Fig. 2). The parts arriving on the conveyor can have different shapes (e.g., square, rectangle, triangle), each of which requires a different packing strategy. The control system may not initially know how to pack all of the possible types of parts— it may have to perform some search algorithm to derive an appropriate box-packing strategy. However, it is critical that the robot does not allow any parts to fall off the end of the conveyor, so if the robot does not know how to pack an arriving part, it is allowed to simply place the part on the nearby table and proceed with other activities. The robot arm is also responsible for reacting to an emergency alert light. If the light goes on, the system must push the button next to the light before a fixed deadline.

The Puma domain thus includes two sources of hard real-time deadlines (arriving parts and emergency alerts) as well as the opportunity to use search-based AI methods to derive part-packing strategies that will improve performance. CIRCA's primary goal in this domain is to always avoid catastrophic failures due to missed deadlines, and to also try to pack as many parts as possible into the box. When viewed from the real-time system design perspective, CIRCA's task is essentially to design, verify, redesign, and implement control systems that meet hard deadlines in the dynamic environment (part arrivals and emergency alerts) and also intelligently deal with changes that are not predicted (new part shapes).

2. THE AUTOMATED DESIGN PROCESS

Fig. 3 shows a flowchart mapping the steps of a traditional control system design process to related portions of the CIRCA approach. Beginning in the upper left of the figure, the designer (human or automated) is given a specification of the system to be controlled. In the case of CIRCA, this specification has three parts: a set of possible initial world states, a set of state transitions that describe how the world can change, and a set of agent capabilities, describing how the agent can change the world. In the Puma domain, each possible initial world state describes the status of the robot, the conveyor belt, the alert light, etc. The state transitions describe the states in which external events may occur, and the new states that result from the events. For example, a state transition is used to describe the possibility of a part falling off the conveyor after some time delay, leading to an unacceptable failure state. The agent capabilities describe the robot's methods for moving parts and pushing the emergency button. The output specification describes the desired behavior; for CIRCA, the specification includes both goals of avoidance (to stay out of some undesirable situations) and goals of achievement (to attain some desirable situations). In the Puma domain, the avoidance goals correspond to parts falling off the conveyor and the emergency alert timing out. The goal of attainment is to pack parts into the box.

The design phase of the process builds a tentative control system; CIRCA builds a reactive control plan using lookahead planning methods similar to STRIPS (Nilsson 1980). The reactions are cast in the form of simple Test-Action Pairs (TAPs) that specify the appropriate control actions for various possible future states of the world. Each TAP implements a set of tests to recognize a particular class of world states, and an action to perform when the system is in that class of states. For example, a simple TAP in the Puma domain might detect when the emergency light is active and the robot is not holding any parts, and initiate an action to push the emergency button, thus cancelling the alert and avoiding the undesirable failure (timeout) state. Deadlines defined by the transitions in the world model are translated into response-time requirements for each TAP that is critical to system safety. Thus, the TAP that responds to an emergency-alert would have to be tested (and possibly activated) frequently enough to ensure that no emergency alert condition is allowed to time-out and result in failure.

The next phase of the design process verifies that the proposed control system can be executed to meet the timing specifications. CIRCA's planner

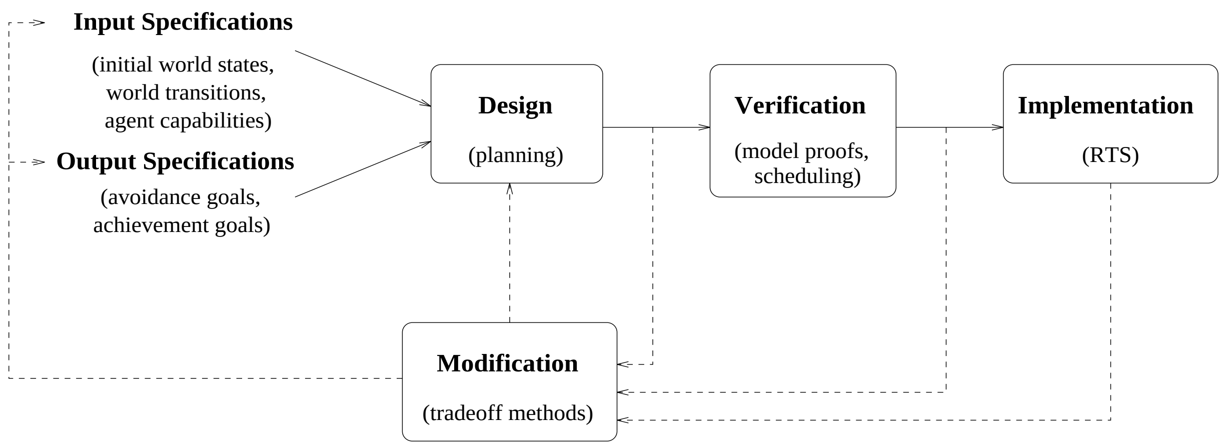


Fig. 3. A flowchart showing the stages of real-time system design.

ensures the logical correctness of a TAP control plan when it is built, based on the world model. The Scheduler reasons about the worst-case execution times of the TAPs, and the limited execution resources available on the RTS, attempting to build an execution schedule that meets all of the response-time constraints[1]. By developing a TAP schedule that meets all of the hard deadlines in some region of interactions with the environment (i.e., some portion of the overall system's state space), CIRCA is essentially designing a real-time reactive control system.

Assuming that the Scheduler is able to produce a feasible schedule that meets all the timing requirements, the AIS can send the schedule to the RTS, which will then execute the TAPs in a predictable manner, enforcing the safety guarantees and behaviors specified by the planning system. A primary feature of these scheduled reactive plans is that, because they are designed to meet all of the domain deadlines in some region of the system's state space, they essentially isolate the AI-based planning system from the real-time response requirements of the environment. While the AIS is performing its planning process, the RTS is simultaneously executing the previously-generated reactive control plan, maintaining guaranteed system safety. This unique combination of communicating but isolated AI methods and hard real-time response guarantees is one of the main performance features of the CIRCA architecture (Musliner *et al.* 1993, Musliner *et al.* 1995).

Following the dashed arrows in the flowchart, it is also possible for the design or verification phase to fail, indicating that some modifications must be made to the initial design or the specifications.

[1] Some TAPs are not responsible for meeting hard deadlines, and these are not assigned response-time requirements. Instead, they are labelled as "if-time" TAPs, which can be executed if time and resources remain unused by the guaranteed TAPs.

For example, the Scheduler may find that it is not possible to run the emergency-alert-response TAP as frequently as specified, so it will return a failure message to the AIS. Feedback from TAPs already executing on the RTS may also initiate modifications to the control system being designed. For example, when parts of a new shape arrive, the RTS will send that information to the AIS, which must develop a new control plan that can pack those parts. Such modifications are essential to automating the overall design process, for two reasons. First, because heuristics are used to generate designs, the initial set of proposed TAPs may be impossible to schedule. A mechanism must be available to modify the planning process (or some other system aspect) so that a different design is heuristically generated and tested. Second, because CIRCA is intended to control an autonomous agent with bounded resources, it is not possible to ensure that the agent will always have sufficient resources to accomplish every task that might arise. As a result, CIRCA must dynamically consider how to apply its limited resources to best achieve its goals, possibly by preferring some goals over others, by changing plans, or by making other modifications to the planning process or specifications.

This capability distinguishes CIRCA's approach from a more traditional design process, in which the input and output specifications are fixed. In contrast, CIRCA may have to modify the I/O specifications of its control system design, when faced with resource limitations. For example, if the conveyor belt is moving very quickly, the system may be unable to guarantee that it will both prevent emergency timeouts and avoid dropping parts. In response, CIRCA might have to prioritize one goal over another, or alter the speed of the conveyor. Both of these changes actually modify the problem specification for the control system, rather than just the control system design. The following section discusses two examples of the types of design tradeoffs CIRCA has experimentally demonstrated.

3. DESIGN TRADEOFFS

Suppose the Puma domain is specified to have parts arriving as little as 45 seconds apart, and emergency alerts as little as 50 seconds apart. The AIS will build a plan including the **pickup-part-from-conveyor** and **push-emergency-button** reactions, both of which must be guaranteed to meet certain deadlines. The Scheduler will then be invoked to see if the available RTS resources are sufficient for those tasks. Fig. 4a shows the results of many such plan/schedule iterations, compiled together to represent essentially a performance profile for the overall task. The axes of the graph show different rates at which alerts and parts may arrive, representing different domains. 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 part arrivals.

3.1 Tradeoff Example: Ignoring Potential Failure

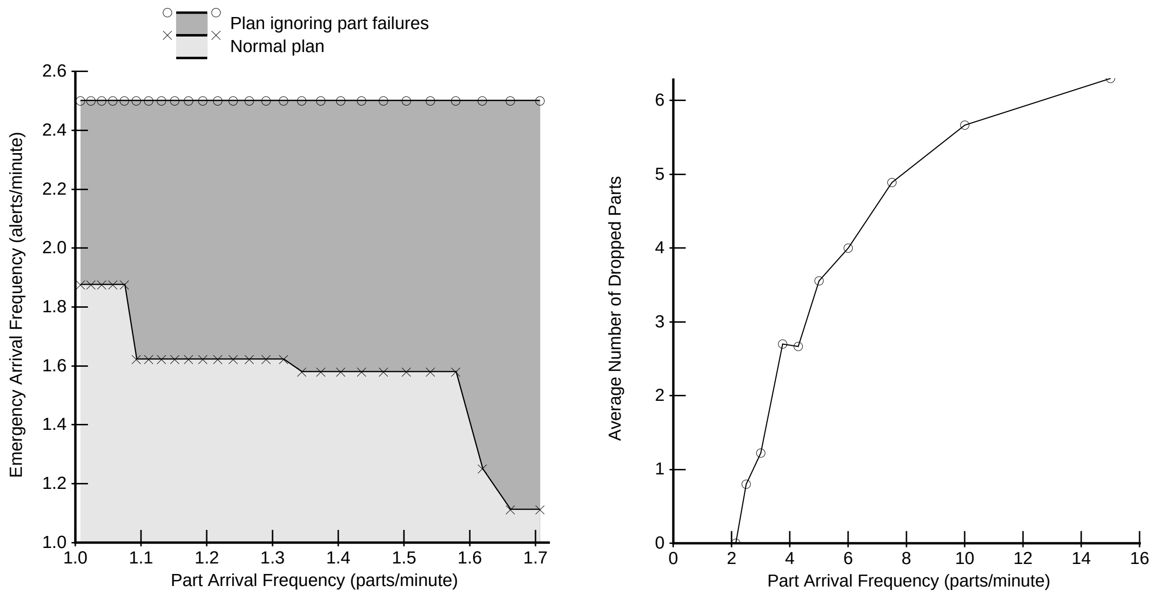
If the arrival rates match a point above the lower curve, then CIRCA cannot build a schedule that will guarantee to avoid both emergency failures and dropping parts. The AIS must make some type of tradeoff to arrive at a plan which is feasible. Suppose, for example, that the system decides that dropping parts off the conveyor is not catastrophic, but merely undesirable. This is equivalent to the system automatically changing the input problem specification so that certain states are no longer considered catastrophic. In that case, CIRCA can build guaranteed schedules for all of the domain 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 system does not need to guarantee the **pickup-part-from-conveyor** action.

To illustrate the non-guaranteed nature of the resulting behavior, this tradeoff method was tested in the Puma domain by increasing the rate of emergency alerts and part arrivals so that the original plan of actions was not schedulable. The AIS then chose to ignore the danger of parts falling off the conveyor, re-planned, and built a new TAP plan in which the **pickup-part-from-conveyor** action was no longer guaranteed, but was instead implemented by an if-time TAP. Fig. 4b shows the expected results: as parts and emergency alerts arrived more frequently, the number of parts falling off the conveyor increased, because the system had less and less free time to apply to if-time behaviors. In this instance, CIRCA traded away its guarantee of preventing parts from falling off the conveyor, in exchange for the ability to guarantee its response to the emergency alert.

3.2 Tradeoff Example: Method Selection

In addition to making changes to the I/O specifications in response to resource restrictions, the AIS can also make changes directly to the implemented form of the planned actions. In particular, the AIS can make changes to the TAPs built to implement action transitions. One powerful modification is to simply alter the specific primitives used to perform the tests and action required by a TAP. The AIS may have several different methods for performing an action (or a test), and it can choose amongst them according to the resources available. This tradeoff method is equivalent to the "configuration selection" (Kuo and Mok 1991), "version selection" (Malcolm and Zhao 1991), and "design-to-time" (Garvey and Lesser 1994) approaches.

For example, suppose that the Puma control system provides the RTS with two different types of part-placement operations, a slow, high-accuracy, "fine-motion" operation and a faster, lower-accuracy, "coarse-motion" operation. This means that the system has two possible primitive operators for the **place-part-in-box** action transition. 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, method selection allows 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). Given the faster coarse-motion operator, the system may be able to guarantee to



(a) Schedulability.

(b) Performance.

Fig. 4. Tradeoffs by ignoring a potential failure.

respond in time to a higher frequency of emergency alerts than with the slower operator.

To provide a more quantitative demonstration of this tradeoff, experiments using these coarse/fine operators were performed. 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.[2] 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 occupy spaces larger than their actual size.

Fig. 5a shows the improvement in response-time achieved by using the coarse-motion operator, illustrated here by the increased rate of emergency alerts and part arrivals that can be handled. The upper curve shows the response tradeoffs that can be made using the faster coarse-motion packing operator, while the lower curve shows the performance for the fine-motion operator used in the previous experiment (and previously graphed in Fig. 4a). The coarse-motion operator reduces the time allocated to the **place-part-in-box** TAP, so the system can respond in time to more frequent part arrivals, emergency alerts, or both.

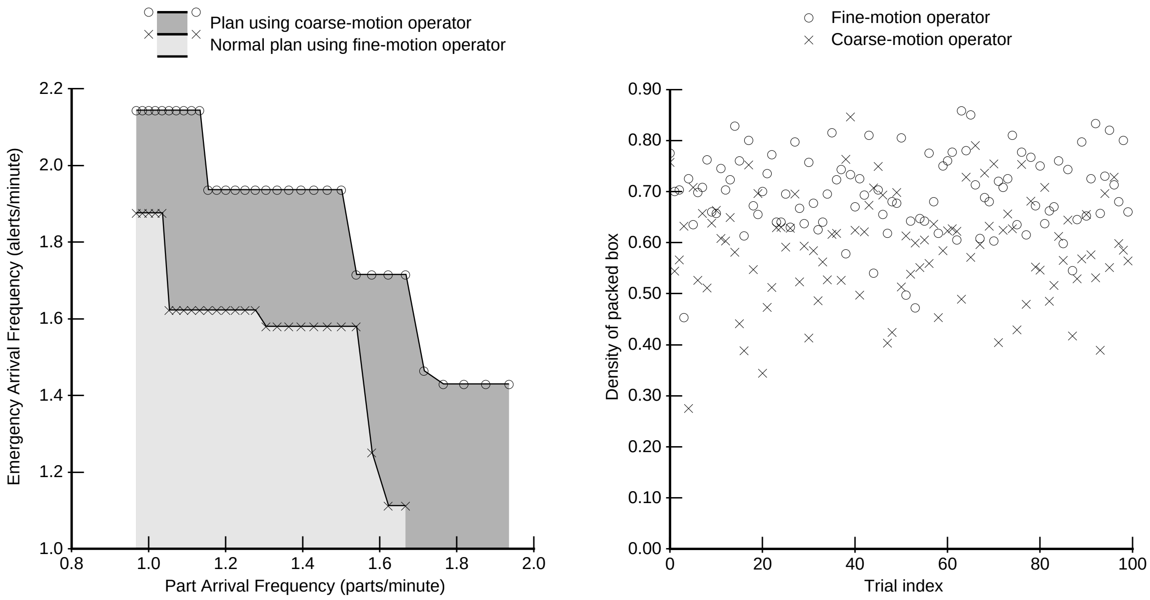
However, Fig. 5b shows the corresponding decrease in performance quality that resulted from the coarse-motion operator, when applied to 100 trials using randomly ordered arrivals of four different part shapes. On average, the density of the packed box was reduced from 70% using the fine-motion operator 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 the improved schedulability and response time illustrated in Fig. 5a are only achieved at the cost of stiff performance degradation.

[2] The box-packing strategy does not deliberately reorder the parts by placing them on the table and packing them later. Parts were only put on the table if the packing operation was aborted to deal with an emergency.

4. CONCLUSIONS

In summary, by automating the entire process of designing and implementing reactive real-time systems, CIRCA is able to intelligently adapt its behavior while still meeting hard real-time deadlines. The view of CIRCA as an automated design system highlights the importance of its mechanisms for making the tradeoffs that are inevitable in resource-bounded real-time systems. CIRCA derives a high-quality plan based on its model of the world, then checks to see if the plan's resource requirements are feasible. If not, the system has many alternatives for making tradeoffs, including sacrificing its guarantees of avoiding particular types of failures, and degrading other performance qualities in exchange for schedulability.

The experiments described above have demonstrated these tradeoff methods. It is important



(a) Schedulability. (b) Density of packed box.

Fig. 5. Tradeoffs by using different TAP implementations.

to realize that, because CIRCA can reason internally about the effects these tradeoff methods have on its performance, the system is "aware" of the tradeoffs it can make, and can choose the effects that are most suited to its overall goals. One of the major directions for future research is the determination of the precise conditions to which each of the available tradeoff methods is best-suited.

5. REFERENCES

Garvey, A. and V. Lesser (1994). A survey of research in deliberative real-time artificial intelligence. *Journal of Real-Time Systems* **6**(3), pp. 317–347.

Kuo, T.-W. and A. K. Mok (1991). Load adjustment in adaptive real-time systems. In Proc. Real-Time Systems Symposium, pp. 160–170.

Malcolm, N. and W. Zhao (1991). Version selection schemes for hard real-time communications. In Proc. Real-Time Systems Symposium, pp. 12–21.

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

Musliner, D. J., E. H. Durfee and K. G. Shin (1995). World modeling for the dynamic construction of real-time control plans. To appear in *Artificial Intelligence*.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga Press, Palo Alto, CA.