Development of Iterative Real-time Scheduler to Planner Feedback[1]

Charles B. McVey Ella M. Atkins Edmund H. Durfee Kang G. Shin

University of Michigan
139 ATL, 1101 Beal Ave.
Ann Arbor, MI 48109-2110
{mcvey, marbles, durfee, kgshin}@umich.edu

Abstract

Planning for real-time applications involves decisions not only about what actions to take in what states to progress toward achieving goals (the traditional decision problem faced by AI planning systems), but also about how to realize those actions within hard real-time deadlines given the inherent limitations of an execution platform. Determining how to arrange actions in a sequence such that timely execution is guaranteed within constraints is a manifestation of the scheduling problem. All cases of the scheduling problem in any domain of non-trivial complexity are difficult to solve (NP-Hard). To more efficiently solve the real-time plan scheduling problem, we propose and analyze an iterative feedback/constraint relaxation method in which a scheduler and planner iteratively interact to efficiently develop a well-utilized schedule which includes as many planned actions as possible. This method has been successfully implemented within the Cooperative Intelligent Real-time Control Architecture (CIRCA).

1 Introduction

Generating plans for the control of a real-time system is an extension of the traditional AI planning problem. Actions must be determined to guide the system from one state to the next, eventually reaching the goal state, as in standard planning. However, unlike traditional planning, these actions are time and resource dependent: they must be executed subject to the limitations of a particular execution platform within hard real-time deadlines to assure that the system is successful in achieving its goals. Plans need to fit the abilities of the execution system.

From the real-time perspective, this "fit" means that the demands of plan execution be schedulable on the system. Scheduling is an NP-hard problem, requiring that the scheduler have knowledge about what the system being scheduled can and cannot do. Meanwhile, the planning process (which is also NP-hard) attempts to generate demands on the system that will accomplish goals. A modular, agent-oriented approach to the overall problem is to couple separate planning and scheduling components, where each applies its own expertise and together they allow the system to achieve its goals reliably within its inherently bounded range of capabilities.

Decoupling planning from scheduling cannot be complete, however. Viewed as a configuration task [Stefik, 1995], it is not the case that the selection of the component pieces of the plan can be done independently of trying to arrange them within the constraints of the execution system. More generally, the problem requires iteration between developing alternative plans and evaluating the schedulability of those plans, until an executable plan that maximally accomplishes goals is found. The obvious question, then, is what knowledge should be passed between these component agents during this iteration to guide the search into promising areas.

In this paper, we detail the development of scheduler feedback mechanisms intended to support the iterative formation of real-time guaranteed control plans. Unlike prior work in this field [Garvey *et al,* 1994] we propose a cooperative protocol in which a scheduler makes state-space search modification suggestions to the planner as opposed to presenting multiple schedules for acceptance based on various criteria. Iterative scheduler/planner feedback as described in this paper is generally applicable to any system which can be mapped to a planner/scheduler agent-oriented model. We have implemented and tested our feedback mechanisms in the context of a particular system, the Cooperative Intelligent Real-time Control Architecture (CIRCA) [Musliner *et al.,* 1995], applied to automated aircraft control in flight simulation [Atkins *et al,* 1996], a domain which demands strict real-time response.

'This research was supported under NSF Grant IRI-9209031.

2 Iterative Real-time Planning/Scheduling

The process of planning can be thought of in three distinct stages: projecting combinations of modeled features forward in time to find reachable states, selecting actions to manipulate this set of reachable states, and determining constraints on those actions, such as timing requirements, necessary to ensure desired changes to reach a goal state or avoid failure states. Real-time execution, on the other hand, deals with determining the current state of the system, finding an action for the system to take (if any), and executing the action within timing constraints.

Determination of the reachable system state set involves expanding a combinatoric search space, however, and is infeasible for a real-time system to detect and react to each state. Instead, since there are typically far fewer actions than states, each expanded state can be classified by the particular action which should be taken in that state. The real-time system may then execute a single task for each *action* rather than a task for each *state.*

Since these actions are typically time dependent, meaning that execution must be completed before a certain deadline to guarantee system transition into the new desired state, tasks must be explicitly allocated resources on the execution platform. In most real world domains, the set of tasks often requires more resources than are available, forcing the system to either fail or consider trade-offs.

The process of making trade-offs must be done carefully to cause sufficient pruning of tasks to make scheduling feasible while avoiding over-extensive pruning, which causes under-utilized schedules and sub-optimal goal achievement. To minimize the risk of over or under pruning the task space, the knowledge of both the scheduler and planner must be brought to bear. We propose a method of iterative negotiation in which the planner first generates its "best" plan in terms of accomplishing goals. The scheduler then schedules the plan if possible, otherwise informing the planner that a new plan must be tried. The process repeats as necessary until a successful schedule is constructed.

An iterative scheduler feedback protocol must specify what information will be contained within request and feedback messages. This depends on the division of knowledge maintained between the planner and scheduler. The planner is an expert at determining which tasks must be performed subject to which constraints to solve the global problem at hand, while the scheduler is an expert at manipulating the tasks into a specific order such that constraints are not violated. Ideally, one would like the scheduler to know *only* how to manipulate tasks into a sequence which does not violate constraints, while a planner knows about the global problem at hand and the tasks required to solve the problem, but *not* the details of how to organize the tasks into a schedule. For scheduler feedback to work effectively, however, the two must share some knowledge. How much knowledge should be shared and how to represent this shared knowledge is not clear. It is undesirable and impractical for the planner to share everything it knows about the global problem with the scheduler [Garvey *et al.,* 1994], and vice versa.

The major question which remains, however, is the exact nature of the feedback provided by the scheduler such that the planner's search is guided rather than relying on extensive blind generate and test cycles. If the planner were allowed to consider schedulability constraints during the process of planning, the question of feedback would be of no concern since only schedulable plans would be generated. However, all advantages of modularity would be lost, and the solution obtained would likely be sub-optimal due to the impracticality of conducting the exhaustive search required to find a well-utilized schedule during planning.

Alternatively, the planner could allow the scheduler to automatically explore variations of the task request, returning the best possible schedules for pre-defined criteria [Garvey *et al.$_y$* 1994]. However, dropping, adding, or changing the timing of a task could change the whole topology of the reachability graph, creating the need for increasing and/or decreasing the importance of many other tasks. This would be acceptable if the planner identifies and indicates tasks that are nearly independent and preferable.

Finally, the scheduler could provide feedback to the planner which actually guides the search of the planner. This feedback would suggest how much less (or more) should be demanded in the task request list. Since it has knowledge of excess available resources or particular conflicts which cause infeasibility of scheduling, the scheduler is in a good position to base such a suggestion upon this information. This final approach has been taken in our implementation within CIRCA.

3 CIRCA Background

CIRCA's realization of real-time AI emphasizes allowing the planning algorithms to be intelligent *about* real-time rather than forcing them to be intelligent *in* real-time [Musliner, 1995]. CIRCA's approach is achieved by separating the architecture into three distinct modules (Figure 1): the Planner, Scheduler, and the Real-Time Subsystem (RTS). The Planner includes a domain-specific knowledge base and a planner which generates 'Test-Action Pairs" or "TAPs" analogous to the tasks discussed above. These TAPs are constructed based on transitions, goals, and actions modeled in the knowledge base.

The Planner begins with a known (set of) initial state(s) and searches a discrete (feature, value) paired state space via modeled transitions in a best-first (descending probability) manner. As the search progresses, each state is assigned a probability calculated from the probabilities of its ancestors

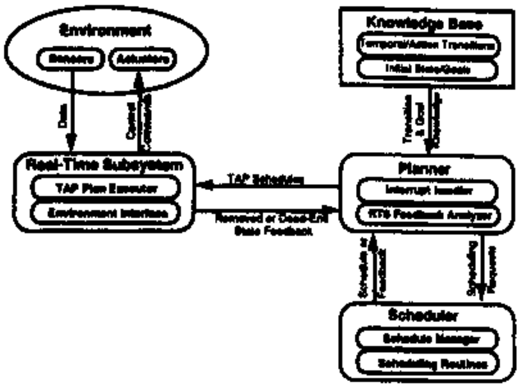

Figure 1: The CIRCA system.

and transitions leading to the state [Atkins *et al.,* 1996]. A cut-off threshold probability is used to limit state expansion. Actions are chosen by the planner to preempt any temporal transitions to failure, and hard real-time "required" TAPs are developed for these actions. "If-time" TAPs are also constructed to pursue non-critical goals specified in the knowledge base. These TAPs do not require real-time constraints since they need not be guaranteed to preempt transitions to failure.

Once the planner builds a complete list of required and if-time TAPs, the scheduler attempts to form a packed (no idle time) periodic schedule in which each required TAP is executed at least fast enough to meet its deadline. A successful schedule, which guarantees failure avoidance to the level of the probability cutoff threshold, is then transferred to the RTS, where it is executed until a new schedule is available. If scheduling is unsuccessful, an iterative process of feedback and re-planning is begun until a successful schedule is developed.

3.1 The Test Action Pair (TAP)

A TAP is implemented as a class within the scheduler with the following fields: test, action, worst-case-execution-time, separation-constraint, and utilization. The test and action slots contain strings which specify execution functions. The execution of a TAP involves first evaluating the test, which if satisfied, causes the paired action to be executed as well. The worst-case-execution-time (wcet) is the time that both the test and action together require to complete *in the worst case.* CIRCA builds plans and schedules based on worst case execution times to make real-time guarantees.

A separation constraint, similar to a period in periodic scheduling literature [Liu and Layland, 1973] but subtly different, is calculated for each TAP which is to guarantee failure avoidance. Critical transitions to failure are modeled to occur with a minimum delay time of D seconds, therefore it is only necessary that a TAP designed to preempt the failure execute *at least once* every D seconds, *not* precisely every D seconds (periodic) [Musliner, 1995].

The utilization of a TAP measures the minimum fraction of CPU time the TAP requires. It is defined as the ratio of its worst case execution time to its maximum separation constraint. The scheduler uses this information to determine if scheduling is certainly infeasible before attempting any scheduling, or to determine appropriate feedback in the event that scheduling is unsuccessful.

3.2 The If-Time Server

Using techniques from [Musliner *et* a/., 1995], we have implemented the scheduling of an if-time TAP server to use slack resources available in a schedule. When executed, this server executes if-time TAPs in a prioritized fashion.[2] The use of the server is preferrable to individually inserting if-time TAPs into the schedule because the scheduler would require explicit domain-specific knowledge about the priorities of the less critical if-time tasks. Instead, when the schedule is executed on the RTS, free time gained when actions require less then their worst case execution times is distributed among if-time TAPs using whatever scheduling policy the if-time server employs. The wcet of the server accounts for this TAP selection time plus the largest wcet of any if-time TAP, thus the server is a guaranteed task.

Since the server does not preempt a transition to failure, it does not have an a-priori separation constraint. To insert it into the schedule as tightly as possible, a binary-like search is conducted through the server's possible utilization space. The utilization of the server can range from zero through one minus the total utilization of the required TAPs. The initial utilization for the first search iteration is simply set to the average of these upper and lower bounds. As an additional mechanism to aid rapid convergence, the search itself is not quite binary: a factor equal to one minus the utilization of the required TAPs is used to partition the search intervals, whereas a binary search would use a factor of 0.5. In testing with 2200 random scheduling requests, this heuristically aided binary search converges faster than or as fast as binary search in 51% and 27% of the cases respectively.

4 Feedback Scheduler Design

4.1 The Base Scheduler

Schedules are built based on a *separation constrained* method of scheduling described in [Musliner, 1995]. The scheduler simulates the execution of a dynamic scheduler by maintaining a time counter and iteratively incrementing it as TAPs are chosen for execution. At each iteration, the TAP with the shortest slack time is initially chosen to be executed. The slack time is defined as the difference between the TAPs separation constraint and the current time minus the time when the TAP was last chosen for

[2]Currently, priority is given to the least-recently executed if-time TAP, yielding a round-robin strategy.

execution:

$$TAP_{Slack\ Time} = TAP_{Sep\ Const} - (Current\ Time - TAP_{Last\ Run})$$

If any other TAP (TAP) can execute within the slack time of the originally chosen TAP:

$$TAP'_{WorstCaseExecTime} < TAP_{Slack\ Time}$$

it will be selected for placement in the schedule instead. If the slack time of any TAP is less than zero at any point, the TAP's deadline is violated and scheduling fails.

After all TAPs are present in the schedule, the scheduler continues its simulation until a valid periodic subsequence containing all TAPs can be extracted as the final schedule.

4.2 The Schedule Manager

A scheduler capable of providing meaningful feedback must have authority to manipulate and retry scheduling the requests it receives from the planner. Given this capability, the scheduler can use the difference between a satisfiable request and over-constrained request to provide more accurate feedback to the planner.

We have augmented the original CIRCA Scheduler with a new rule-based system known as the Schedule Manager (or "Manager") which directs the processing of all Planner scheduling requests. Depending upon the request, the Manager may perform a variety of actions: schedule a request, modify some constraints in a request, modify parameters which govern behavior of the core scheduling algorithm, calculate appropriate feedback, and transmit a valid schedule or feedback.

After each scheduling attempt on a request, the Manager invokes rules which determine what should be done next based on the result code(s) returned from the attempt, any error conditions, instructions received from the Planner, and scheduling strategies in the rules. The result codes are:

- SCHEDULE-WITH-SERVER
  The original scheduling request from the Planner with the if-time server was successfully scheduled.
- SCHEDULE-TOO-LAX
  The schedule found is under-utilized.
- SCHED-NO-SERVER
  The required TAPs specified by the Planner were successfully scheduled, but the if-time server could not be scheduled.
- PARTIAL-SCHED-WITH-SERVER
  Some of the required TAPs specified by the Planner were successfully scheduled, along with the server.
- PARTIAL-SCHED-NO-SERVER
  This result is the same as the previous one, except no if-time server could be inserted into the schedule.
- NO-SCHEDULE
  The Planner's original scheduling request could not be scheduled, and no relaxations were allowed.
- NO-PARTIAL-SCHEDULE
  The Scheduler could not satisfy either the Planner's original request, or any subsequent legal relaxed request.

Failure to generate a schedule which meets the original request from the Planner will generate an error condition which indicates the specific constraint violation that occurred, along with TAP(s) which caused the violation. The Manager then returns either a suggested probability threshold or relaxes constraints and tries scheduling again.

A new probability threshold recommendation is made based on a heuristic-guided binary search (similar to that described earlier for if-time server scheduling) between the minimum, maximum, and current threshold used by the Planner. It is calculated using the priorities and utilizations of recently attempted schedules. When the Planner adopts an increased probability threshold, the state search space is pruned, causing the separation constraints of the TAPs to be increased and/or the removal of some TAPs from the scheduling request. Either of these effects trades off some degree of system completeness for reductions in the difficulty of constructing a feasible schedule. A decrease in the threshold has the opposite effect, causing more states to be expanded, smaller TAP separation constraints, and possibly more TAPs in a scheduling request, increasing system response capabilities and scheduling difficulty.

If the rule invoked by the Manager instead suggests relaxing the constraints on the current scheduling request and trying again, one of the two methods discussed below will be employed.

Trading Off the WCET of the If-Time Server

Initially the execution time of the if-time server is defined as the maximum of the if-time TAP execution times. While this guarantees that if the server is scheduled, each if-time TAP will have a chance to execute, it is an extremely restrictive choice, forcing the scheduler to either allow all possible if-time TAPs (when the schedule was successful) or none (when the schedule was not successful).

If a conflict occurs between the server and a required TAP, the Manager can reduce the execution time of the server to the greatest if-time TAP wcet value possible which removes the conflict. Chances are still good that time will be available to execute any if-time TAPs precluded from the server since schedules are built using worst case execution times.

Priority Scheduling

Another way of incrementally relaxing the request from the Planner is to selectively remove required TAPs from the request A schedule request with the TAP combination of the next highest total priority is attempted. Differences between the original request and a successful partial schedule are used by the Scheduler to provide more accurate feedback to the Planner. The Manager does not require further instruction from the Planner during this process.

The priority of a TAP is originally set by the Planner to be proportional to the average of the probabilities of the world states from which that TAP might be executed. These priorities serve as an approximate representation of the Planner's search space, isolating the full complexities of the domain from the Scheduler.

4.3 Scheduler/Planner Messages

The Planner transmits scheduling requests consisting of TAP data and instructions for making scheduling trade-offs to the Scheduler. The Planner can currently select among two different primary instructions to the Scheduler:

- SCHEDULE-THESE-TAPS
  This instruction is followed by parameters which govern the scheduling process and scheduling data about the TAPs. Data for each TAP may include worst case execution time, separation constraint, and priority.
- SCHD-PREV-TAPS-NEW-PARAM
  This instruction is followed by the new scheduling trade-off directives only.

The parameters which the Planner may specify include:

- IF-TIME-SERVER
  This parameter allows the Planner to specify whether insertion of an if-time server into the schedule to be built is *required, desired but not imperative,* or *not useful.* This gives the Planner the capability of trading off the need for an if-time server if resources and/or deliberation time are constrained. The Planner will usually *require* its insertion, unless "quick" replanning is required to respond to some emergency [Atkins, 1997].
- TRADE-OFF-SERVER-EXEC-TIME
  This boolean indicates whether Scheduler trading-off of the server worst case execution time is permitted.
- LEVELS-OF-PRIORITY-SCHEDULING
  This parameter specifies how many different priority levels the Scheduler may analyze before aborting.

The Planner also sends the Scheduler three probabilities: the maximum and minimum probabilities of the set of expanded states and the current threshold probability. Note that the current threshold probability is always less than or equal to the minimum expanded state probability. These probabilities are used during the Manager's binary search, as discussed in Section 4.2.

The Scheduler will return either a new suggested probability threshold or a successful schedule. A new threshold message can occur either when the Scheduler is over or under constrained: the new value being either greater than or less than the previous threshold, respectively.

5 Testing Scheduler Feedback

Extensions were introduced into the previous simulated flight domain knowledge base [Atkins *et* a/., 1996] to model more potential dangers, forcing the planning and scheduling of more preemptive TAPs. The addition to the knowledge base consisted of modeling the possibility of colliding with other air traffic *at any point in the flight.* Traffic was modeled in the system through the use of three actions (AC): a void-traffic, course-correct, and resume-heading. These actions are designed to preempt temporal transitions to failure (TTF) and rely on temporal transitions (TT) to function correctly together (Figure 2).

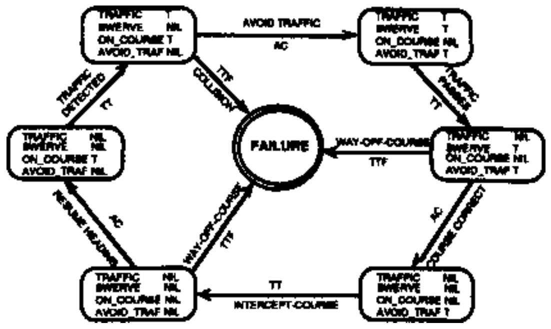

Figure 2. CIRCA traffic avoidance world model.

This complex chain of events requires the Planner to request scheduling of three additional TAPs. One of the Planner-Scheduler iteration cycles generated during the operation of CIRCA is presented for analysis. The original scheduling request sent from the Planner to Scheduler is shown below.

Instruction: SCHEDULE-THESE-TAPS
Current/Min/Max Probability Threshold: 0.0/0.0/1.0
LEVELS-OF-PRIORITY-SCHEDULING: 5
IF-TIME-SERVER: Required
TRADE-OFF-SERVER-EXECUTION-TIME: No

TAP INFORMATION

TAP #	WCET	Sep.Const	Priority
0	2150	45000	38
1	2150	45000	41
2	4150	9000	15
3	2150	20000	47
4	5325	90000	41
5	3550	N/A (server)	N/A (server)

The initial attempt to schedule all required TAPs (0-4) failed because of a conflict between TAP 2 and TAP 4. Since priority scheduling was requested, the Manager decided to remove TAP 2 (the lowest priority TAP) from the request and try again. In the second scheduling iteration, the schedule: {3, 1, 5, 4, 5, 3, 0, 5} was constructed, with the if-time server separation constraint bound to 9628 after the binary search. A rule within the Manager specifies that if a partial schedule is successfully made, the Manager should calculate a new threshold probability suggestion for the Planner and return it as feedback. In this case, the algorithms for calculating the new threshold generated the value of 0.057, which was returned to the Planner.

The Planner, using the new threshold, replanned and submitted a new request to the Scheduler which was similar to the original request but with slight timing relaxations and *without* the previously troublesome TAP 2. This request was satisfied by the Scheduler, and the successful schedule was returned.

By comparison, before the feedback mechanism was added, a scheduling failure caused the Planner to blindly increment its probability threshold by 0.1 and replan. This resulted in a successful but under-utilized schedule. There was no method for detecting under-utilization and subsequently decreasing the threshold, thus this non-optimal schedule was accepted and the system performed below its capabilities, potentially failing to react in time to a fatal situation which had been needlessly pruned during planning.

6 Future Research

An open question which needs to be addressed is the handling of under-utilized schedules. Tests of the current implementation have shown that in some cases the suggested probability threshold is too high. When this occurs, too many TAPs may be excluded from subsequent requests. To prevent this, the Scheduler (or Planner) must be able to ascertain when a valid schedule is under-utilized.

In the future, the Planner should be provided with the capability to reason about what it should do given Scheduler feedback. This reasoning will likely be domain dependent, and could be specified in the form of production rules which indicate user or designer preferences. Alternatively, the reasoning may borrow from decision theory, computing the expected utility for different courses of action and choosing the strategy which yields the most benefit. Currently the Planner blindly adopts whatever probability threshold suggestion or schedule the Scheduler sends back, which is not an ideal policy given the Scheduler's roughs indirect knowledge of states and their probabilities.

7 Conclusions

We have addressed the difficult problem of interfacing an AI planning system to a real-time scheduler by proposing, developing, and implementing an iterative feedback mechanism. This mechanism allows a large degree of decoupling between the scheduler and planner, enabling the two modules to each perform within its realm of expertise, communicating with mutually meaningful information in a controlled protocol to solve a global problem.

Further refinement of the CIRCA-specific methods for calculating probability threshold feedback and detecting under-utilized schedules is needed. However, our scheduler feedback method gives the system an increased chance of efficiently meeting goals under resource constraints by providing quantitative knowledge to the planner, eliminating the need for blind search. This represents a crucial step towards the realization of a fully self-reliant real-time AI architecture capable of solving difficult real world control problems such as completely automated flight.

References

[Atkins *et al,* 1996] Ella M. Atkins, Edmund H. Durfee, and Kang G. Shin. Plan Development using Local Probabilistic Models. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference,* August 1996.

[Atkins *et al.,* 1997] Ella M. Atkins, Edmund H. Durfee, and Kang G. Shin. Buying Time for Resource-Bounded Planning. To appear in *AAAI-97 Workshop: Building Resource-Bounded Reasoning Systems Technical Report,* July 1997.

[Liu and Lay land, 1973] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Haid-Real-Time Environment. In *Hard Real-Time Systems.* IEEE Press, pages 174-189, 1988.

[Garvey *et* a/., 1994] Alan Garvey, Keith Decker, and Victor Lesser. A Negotiation-based Interface Between a Real-time Scheduler and a Decision-Maker. Technical Report 94-08, University of Massachusetts Department of Computer Science, March 1994.

[Musliner *et al„* 1995] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. World Modeling for the Dynamic Construction of Real-Time Control Plans. *Artificial Intelligence,* vol. 74, pp. 83-127, 1995. American Association for Artificial Intelligence.

[Musliner, 1995] David J. Musliner. Scheduling Issues Arising from Automated Real-Time System Design. University of Maryland Department of Computer Science Technical Report CS-TR-3364, UMIACS-TR-94-118, January 1995.

[Stefik, 1995] Mark J. Stefik. *Introduction to Knowledge Systems.* Morgan Kaufmann, San Francisco, California, 1995.

PROBABILISTIC REASONING

PROBABILISTIC REASONING

Probabilistic Reasoning Distinguished Paper