Plan Execution in Mission-Critical Domains

(Position Paper)

David J. Musliner
Automated Reasoning Group
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
musliner@src.honeywell.com

Introduction

AI planning was born into a simple world: baby's blocks, slow robots, benign and cooperative environments. Now, as a maturing technology, AI planning is facing the real world: satellites, jets, the World Wide Web, and tax law. Moving planners and plan executives out of research labs and into industrial and public applications is difficult for many reasons; my focus here is on the numerous confidence, certification, and reliability issues faced in the real world.

In particular, many of the domains to which planning seems most suited (e.g., autonomous vehicles such as satellites and aircraft) are *mission critical*, in that system failures can lead to catastrophic results: loss of life or huge costs. Traditional aircraft and satellite control systems are extensively tested and certified to behave in very well-understood ways. Substituting an AI planning system and plan executor for a Fortran satellite control routine will take a lot of convincing.

In this paper, I take the position that the best way to address these issues is through the use of an inherently self-validating planning and execution system. That is, a system in which the planner generates plans satisfying certain verifiable properties (e.g., timeliness and correctness), which are then predictably and reliably executed by the executive. In abstract form, this type of system is essentially a high-level automatic programming paradigm: the system designer provides a description of primitive sensing and control actions, a description of the domain and its dynamics, and a description of the system's goals. Then, conceptually at least, the system generates and executes a plan composed of primitives and combination functions (control logic) to reliably achieve the goals. Once the system code itself has been certified, the only further verification/certification requirements apply to the input models of primitives and the domain; each plan (program) is itself verified automatically during generation.

The popularity of such classical planner/executor architectures waned during the 1980s as reactive systems dawned, but more recently the community has almost converged on multi-layer architectures incorporating planners with reactive plan execution engines. In the transition, however, many of the advantages of the classical architecture have been lost: some planners now "advise" rather than "program," and execution engines themselves are frequently running complex, hand-coded, unbounded skeletal/hierarchical plans. These systems do not provide the advantages of the "automatic programming" paradigm.

So, let this position paper act as a **call to arms**: bring back predictability, execution semantics, plan verification and other such features. Stem the rampant growth of plan executor features in favor of developing accurate models of how the existing systems actually work, so that projective planners can build plans rather than simply turning on and off human-built plans. With a more rigorous understanding of the performance features of complex plan execution systems like RAPs (Firby 1987), their powerful flexibility can be used in the context of verified systems for mission critical domains.

It is important to note that the drive for a more predictable system does not necessarily require us to abandon the recognized advantages of new methods like reactive execution. For example, the Cooperative Intelligent Real-Time Control Architecture (CIRCA) (Musliner, Durfee, & Shin 1993; 1995) roughly follows the automatic programming model, with enhancements designed to handle a variety of realistic resource limitations including both planning time and execution time. CIRCA also incorporates some of the fairly recent advances in planning and execution techniques (e.g., indexical-functional variables (Agre & Chapman 1987), reactive execution). The following sections provide more detail on how CIRCA's plan execution component uses these techniques while also providing a strictly predictable execution behavior that CIRCA relies on to enforce guaranteed plans for mission-critical domains.

The CIRCA Approach

In the CIRCA approach to automated control for mission-critical domains, the planner reasons about a fairly complex model of world dynamics and primitive control behaviors (Musliner, Durfee, & Shin 1995). Projecting the world dynamics forward, the planner synthesizes plans in a quite simple form called a Test-Action Pair (TAP) schedule, which is essentially a loop of if-then rules that are sequentially tested and executed. The simplicity of this structure for plans is crucial, because the planner must have a complete understanding of the logical and temporal consequences of each TAP it plans. Through projection, the planner reduces decisions about parallel goal pursuits, goal conflicts, and other complex issues to simple, state-based reactions that do not require the plan execution component to perform any significant multi-step inference, projection, or conflict resolution. The planner's world model incorporates indexical-functional state features explicitly, thus realizing both their advantages for efficient planning (Musliner 1994) and reactive execution.

By the time the plan gets to the executor, its behavior must be very well understood; at a minimum, the planner must have a certain level of assurance that the plan is correct. CIRCA's concept of a "safely-controlled set of states" supports this level of assurance. The planner works to derive plans that restrict the world to a set of states satisfying two basic properties:

Safety — No catastrophic failures are reachable within the set.

Closure — The (controlled) world state will remain within the set of states until the system is ready to leave the set.

Executing a plan that enforces a safely-controlled set of states allows the CIRCA planner to work indefinitely to find a plan for the next phase of operations, while the current plan keeps the system safe.

Focusing on the executive component, this approach places several unusual requirements on the functional behavior of the system. For example, completely predictable execution of the plan is required to ensure that the planner's intentions are accurately carried out. This predictability must span both logical behavior (i.e., the executive should take exactly those actions planned by the planner for any given state of the world) and timeliness (e.g., the reaction time of the executive to changes in the world must remain strictly within the bounds established and expected by the planner).

CIRCA raises the timeliness aspects of plan execution to the same level of concern as the logical correctness standards associated with traditional planning. Timeliness involves not just reasoning about time at a coarse level during plan generation, but also detailed timing information that explicitly accounts for sensing activity, the delays between sensing and action (Musliner, Durfee, & Shin 1994), communication delays, and the lowest-level details of action selection and execution.

CIRCA's Predictable Executive

Building this type of predictable executive for the CIRCA system required several essential design elements. Reviewing these design decisions is made particularly interesting by keeping in mind the features of the more common executives such as RAPs (Firby 1987) and PRS (Georgeff & Ingrand 1989; Ingrand, Georgeff, & Rao 1992). In CIRCA, the Real-Time Subsystem (RTS) fulfills the role of plan executor, taking in cyclic loops of Test Action Pairs (TAPs) from the parallel planning system. TAPs are essentially automatically generated/planned if-then production rules annotated with worst-case resource and timing requirements. Designed to meet mission-critical hard real-time performance criteria, the CIRCA RTS includes[1]:

Scheduled Communication — All communication into and out of the RTS is explicitly scheduled within TAPs. That is, all communication actions are actually accomplished from within the planned TAPs. This includes incremental downloading of new TAPs schedules from the planning/scheduling system, and sending feedback information about world state and plan execution progress back to the parallel planning system.

Scheduled Sensing — All sensing activity is likewise encapsulated within TAPs, and thus explicitly scheduled onto the available sensing and processing resources.

Scheduled Dynamic Memory — Similarly, dynamic memory allocation and de-allocation are managed explicitly within the planned and scheduled TAPs, and are thus both predictable and closely controllable. As with communication, allocation functions are performed incrementally, to avoid the costly worst-case of bulk allocations. In the current implementation, dynamic memory allocation is used during downloading of a new TAP schedule (plan), but nowhere else; appropriate use of indexical-functional variables avoids the need to "gensym" new names for the changing objects in the world.

[1] Complete details on this design are available in (Musliner 1993).

No Interrupts — Unlike many systems which attempt to ensure real-time performance through rapid response to interrupts, the RTS does not accept any interrupts. Instead, the TAP plan being executed by the RTS is expected to have TAPs that explicitly check for all important conditions as quickly as necessary. In a sense, we have moved the polling loop out of the interrupt hardware and into the software RTS, so that the planner and scheduler can reason explicitly about the form and frequency of that loop. This design greatly increases the architecture's ability to control and predict the responses of the system. If the RTS accepted interrupts, it would be very difficult to make any guarantees about its performance, since the system would have to account for the many unpredictable aspects of interrupt-driven systems. For example, lower-priority interrupt handlers could never be guaranteed, since higher-priority interrupts would preempt and override their behaviors. Furthermore, incoming interrupts might be lost if they arrived during the handling of equal- or higher-priority interrupts.

Summary

We need plan execution systems that are predictable enough to be modeled accurately by planning systems, yet powerful enough to manage the vagaries of the real world. CIRCA's RTS is a very simple first step in that direction.

References

Agre, P. E., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proc. National Conf. on Artificial Intelligence*, 268–272. Morgan Kaufmann.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *Proc. National Conf. on Artificial Intelligence*, 202–206.

Georgeff, M. P., and Ingrand, F. F. 1989. Decision-making in an embedded reasoning system. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, 972–978.

Ingrand, F. F.; Georgeff, M. P.; and Rao, A. S. 1992. An architecture for real-time reasoning and system control. *IEEE Expert* 34–44.

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

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1994. Predictive sufficiency and the use of stored internal state. In *Proc. AIAA/NASA Conf. on Intelligent Robots in Field, Factory, Service, and Space.*

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Musliner, D. J. 1993. *CIRCA: The Cooperative Intelligent Real-Time Control Architecture.* Ph.D. Dissertation, University of Michigan, Ann Arbor. Available as University of Maryland Computer Science Technical Report CS-TR-3157.

Musliner, D. J. 1994. Using abstraction and nondeterminism to plan reaction loops. In *Proc. National Conf. on Artificial Intelligence*, 1036–1041.