AIAA-94-1290-CP

Integrating Delibrative Planning in a Robot Architecture

Chris Elsaesser and Marc G. Slack
The MITRE Corporation
AI Technical Center
7525 Colshire Drive, McLean, Virginia 22102-3481
chris or slack @starbase.mitre.org (703) 883-6563

Abstract

Planning researchers are coming to accept that planning is not sufficient for robotics. But many of them seem to think that the concrete levels of robot control which a planner must interface are somehow uninteresting or unimportant to the development of a theory of autonomous control. On the other hand, some robotics researchers appear to think that planning is not even necessary for robotics, that reactive control will be adequate for any realistic robot system. Our thesis is that progress toward autonomous robotics will be stunted until both camps understand that the other plays an equally important role in the creation of non-trivial intelligent autonomous agents. This paper describes the role of planning and reactive control in an architecture for autonomous agents (robots). We posit this is necessary and sufficient. The key to our architecture is the interjection of a "sequencing layer" between the reactive controller needed for a robot to survive in a dynamic environment and a deliberative planner needed to develop a course of action to achieve high-level user goals.

1 Past: Robotic control as a domain for planning

Controlling an autonomous robot is mentioned in many planning research papers as a typical domain. But few planning researchers discuss, or even appear to understand the practical problems of robotic control. The following is the prototypical answer to complaints of the more practically minded robotics engineer: "I have captured the essence of the robot planning problem with a representative problem in which it is easier to present a discussion of all the important issues. The details are unimportant."

That justification would be acceptable if there were evidence that the details are really unimportant. But we know from the recent spate of papers on the complexity of planning that making planning practical are in the details. Abstractions into a "representative problem" obscures the fact that classical AI planning can only address one segment of the robot control problem.

Robotic control from the planning researcher's perspective is often viewed as the task of moving from one location to another [20, 16, 3, 15, 24, 8, 14]. The desired plan would be a sequence of movements for the robot to carry out. Such movement plans usually assume sensing systems capable of generating accurate metric models of the portion of the world relevant to the navigation task. When people got around to building robots that could benefit from this type of planning, they found that nearly all of the interesting behavior could be accomplished more efficiently by the mechanisms assumed by the planner.

The rasion d'etre of deliberative planning is the need to reason about preconditions. It is not too difficult to construct practical examples of why this is necessary for an autonomous agent. If you have more in mind for the robot than getting from here to there, you can usually convince yourself that you need to think about it before "heading in the right direction" and relying on reactivity. Dealing with complications like another agent loose in the environment or the need to do something like repair a broken device (so you need to gather the right tools before you leave) are not activities a purely reactive system can accomplish any more efficiently than a classical planner can accomplish movement planning in unstructured environments.

Chapman proved that planning is computationally intractable and could not possibly be a sufficient theory of intelligent behavior in a real-time environment [5]. Others have catalogued some of the behaviors one would need in an autonomous agent: reaction to unforeseen events, iterative actions (e.g., traveling down a street stopping for all the red lights), real time projection and conditional commitment to action (e.g., cross a busy highway [19]).

Chapman was the vanguard of the "situated reasoning" approach to intelligent agents [4, 1, 6, 17]. These "reactionaries" started at the opposite end of the control continuum from classical AI planning, bent on showing that planning may not even be necessary for intelligent behavior. But the problems the reactionaries cannot address turned out to be important to producing an autonomous agent. Among the more obvious are the inability to employ predictive reasoning about preconditions (check the gas gauge before embarking on a long car trip, don't wait until the car starts sputtering), coordinating activity with other intelligent agents (pick up a heavy object together), reason under uncertainty and take risk-alleviating actions (move into the right lane well before your exit).

Our hypothesis is that planning is necessary for control of autonomous robots, just as situated reasoning or reactive behavior is necessary. The main issues are mediating between deliberation and reaction to produce seamless intelligent behavior, and determining the proper roles of the various components of an intelligent agent architecture.

2 A three-layer architecture for intelligent agents

The robot intelligence community has begun to agree on a software architecture for "intelligent" robotic systems [10, 18, 12, 7, 11, 23, 25]. The consensus emerging is an architecture which incorporates both planning and reaction. In most of the architectures cited, there is a planning component for reasoning about the overall mission and generating contingencies when the mission itself is in danger of failing. Importantly, the planner is asynchronous (but on-line) with a real time reaction component which can achieve the situated reasoning needed for survival and continuous control.

While it now seems obvious there is a role for reaction and planning in robot control, what is not so obvious is how to mediate between the two. Our architecture separates the general robot intelligence problem into three interacting pieces, with the middle piece being the key to mediation between reaction and deliberation (see Figure 1):

1. A set of robotic specific reactive skills. For example, grasping, object tracking, and local navigation. These are tightly bound to the specific hardware of the robot and must interact with the world in real-time.

2. A sequencing capability which can differentially activate the reactive skills in order to direct changes in the state of the world and accomplish specific tasks. For example, exiting a room might be orchestrated through the use of reactive skills for door tracking, local navigation, grasping, and pulling.

3. A deliberative planning capability to reason in depth about goals, preconditions, resources, and timing constraints. The planner generates rough plans for accomplishing goals. For example, given the task to retrieve an item and a map of a building, the deliberative system could reason about the interconnection of spaces and return a plan for the robot to exit the room, follow the hall to the left and enter the third door on the right.

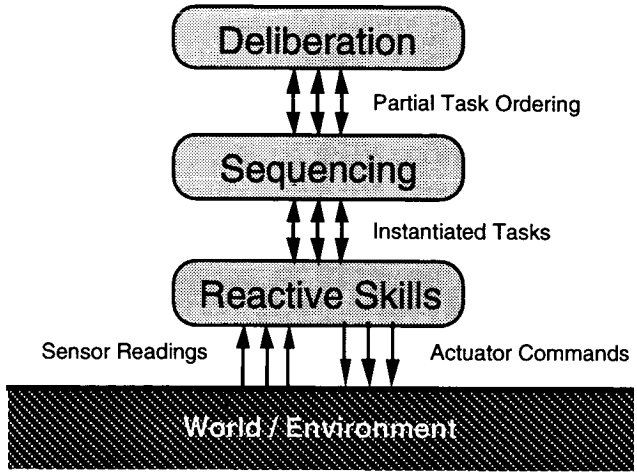

Figure 1: Generic intelligent control architecture

This paper focuses on the interaction of planning and sequencing layers of this architecture. Interaction of the sequencing system with the reactive layer of the architecture is covered in other papers [23, 25].

3 Sequencing: caching techniques for handling routine activities

Reactive control addresses only the most obvious defect of state-based planning for robots, the inability to represent and reason about motor control in real time. But even if provided with primitives such as grasp, track-wall, and so on, a state-based planner will still be unable to efficiently handle everything needed for robot control above the level of reactions that can be compiled into guaranteed-reaction-time primitives.

The most obvious problem is the need to do indefinite iteration of sequences of primitive behaviors:

```
Do unitl (at robot1 door5):
Put-one-foot-in-front-of-the-other
```

The problem with such sequences for a state-based planner is they produce an indeterminate number of states to manage. A plan is basically a proof that some goal can be achieved with a sequence of state transitions effected by atomic plan steps. Planners convince themselves a goal is achievable by constructing the whole plan. Obviously, it is difficult to generate a complete state-transitions plan to embody what we easily expressed by the iterative construct above. That is the reason typical planning representations ground out on primitives like the following:

```
(Operator move
   :purpose (location ?robot ?destination)
   :preconditions ((clearpath ?robot ?destination)
                   ...))
```

Such operators assume things which are hard to encode as states, like keeping away from walls and people. They also assume indefinite sequences of very small grain actions. Situated actions typically implemented in robots often match the state transition view of classical planning perfectly well [13]. The problem is two-fold. One is performance; at the level of abstraction provided by situated actions there would be too many plan steps to generate for even very simple goals like moving from within a room out into a hallway. The second is representation. No one has developed a useful state-based planner that can reason about indefinite iteration of actions and conditional actions.

What is needed is a layer between reactive behaviors and deliberative planning which allows the planner to reason only to the level of routine activities such as move and open-door (grasp, turn, and pull. If that fails, push. If that fails consider another route to the goal). In our architecture, we use Reactive Action Packets (RAPs) to encode routine behavior as a sequence of situated skills [22]. RAPs is a language with a syntax similar to the syntax of classical planning systems [10]. Like most planning systems, the RAP system uses a library of decomposition rules to represent sequences of behaviors to accomplish a task. The system can quickly transform a task into a context specific sequence of primitive actions by caching solutions to common tasks. Unlike a planning system's computationally expensive search mechanisms used to decompose tasks into primitives, the RAP system must have a solution to the given task cached in its library or the system reports a failure. RAPs can encode conditional and iterative sequences of actions since there is no state-based search involved. As is exemplified by the following example, the door is iteratively bashed with a sledge hammer until the not closed state is detected or simply opened if the door is unlocked.

```
(define-rap opendoor
(success (not (closed ?currentdoor)))
 (method
     (context (doorlocked ?currentdoor))
     (t1 (grasp-sledge-hammer) (for t2))
     (t2 (pound-door ?currentdoor)
          (wait-for (not (closed ?currentdoor)))))
  (method
      (context (not (doorlocked ?currentdoor)))
      (t1 (grasp-door-handle) (for t2))
      (t2 (turn-knob) (for t3))
      (t3 (pull-open-door))))
```

While the RAP system can perform task decomposition, it is not suited for direct interaction with the world. The software constructs that are used for selecting action routines, binding variables, and so on make the system too slow for survival. Thus the system is used in our architecture to dynamically configure a reactive layer to handle the interaction with the world for the current task and situation. This allows complex behaviors to be programmed, while relying on always-active situated skills to protect the robot from inaction in a rapidly changing environment.

Sequencing, married to reaction, yields significantly better task coverage than either of the two can provide alone. Still, the combination of the sequencing and reactive layers is not structured to perform complicated resource allocation reasoning. Such is typical in determining the best way to carry out a set of tasks. Nor are these two layers good at reasoning about the failure requirements or consequences of a task. So where the sequencer gains in its ability to handle routine situations (e.g., starting a car, opening and moving through a door), it lacks the ability to string these routine tasks together in a way that will have the desired "global" behavior. But that happens to be just the thing planners are good for.

4 Planning

Our view is that there is a role for state-based planning in robotic intelligence, but it should be limited to tasks that are not easy to specify as sequences of

common robotic skills. When planning is necessary, the planner should think of the problem at the highest level possible in order to make the problem space the smallest possible.

Thus, the role of planning is to deliberate, but only when necessary. The role of reaction is to control real-time behavior. The role of sequencing is to raise the level of abstraction of the lowest level of activities which the planner will concern itself. In the process, eliminate the need for state-based planning of things which are easy to encapsulate in an operator that grounds in an indefinite iteration of low-level skills. Importantly, all three layers must operate concurrently and asynchronously. Accomplishing this is the key to making planning useful in a robot.

Because the sequencer has a cached solution to routine tasks, the planning system has the advantage of building upon this level of abstraction providing it larger grain sized primitives. This eases the complexity of the "planning problem" because it eliminates large numbers of essentially linear planning problems. Ability of the RAP system to deal with iterative behavior greatly simplifies the planner's representation, allowing the simple state representation common to classical planning to suffice in most common situations.

The planner we are using in our experiments is called AP [9]. AP has a number of features which make its role more compelling than robot planning typically exemplified in the planning literature.

One thing missed by both the planning and robot control communities (while they were arguing over the necessity of planning) was autonomous robots generally will be autonomous only from the human giving them orders. They will not often be acting alone in their environment. Multiagent control is necessary when more than one robot is employed to carry out tasks, or when multiple robots are operating independently on multiple tasks in a shared environment.

AP was designed to deal with multiagent coordination. To do this, it extends state-based planning to reason about the conditions that hold during actions. This capability allows AP to plan activities such as two robots carrying a bulky object. The following operator is an example from a test domain. Note the planner can instantiate the variables ?arm-or-robot1 and ?arm-or-robot2 with anything that meets the constraints. A two-armed robot or two single armed robots might be used. The temporal relation "simultaneous" imposes a non-codesignation constraint on the agents so that a very strong one-armed robot would not qualify. Other temporal constraints in the plot language would allow codesignation). These plot temporal constraints also cause the plans steps that instantiate the plot subgoals to include scheduling information that the sequencing layer and AP's execution monitor can use.

```
(Operator pickup-heavy-object
 :purpose (holding ?planner ?large-thing)
 :arguments
    ((?weight-of-thing
       (get-value ?large-thing 'weight)))
  :preconditions
     ((top ?large-thing clear)
      (on ?large-thing ?something))
  :constraints
    ((can-lift ?arm-or-robot1
               (* 0.5 ?weight-of-thing))
     (can-lift ?arm-or-robot2
              (* 0.5 ?weight-of-thing))
  :plot
    (simultaneous
      (grip ?arm-or-robot1 ?large-thing)
      (grip ?arm-or-robot2 ?large-thing))
 :effects
    ((holding ?planner ?large-thing)
     (top ?something clear)
     (on ?large-thing nothing)))
```

Another feature of AP which makes it appropriate for control of autonomous agents is that it can reason about uncontrolled agents. AP was originally developed to address multiagent adversarial domains (AP stands for Adversarial Planner). Of course most robot applications are not adversarial. An uncontrolled agent might be a human operating in the environment along with a robot, or even nature. AP can use its adversarial reasoning capabilities as a risk assessment mechanism to decrease the probability of dangerous interactions with other agents.

AP includes a "counterplanning" component to reason about how an uncontrolled agent might prevent a plan from succeeding, either by negating a precondition or a "during condition." Problems are uncovered and addressed by augmenting the plan with operations which prevent the negative effects of the uncontrolled action. This amounts to reasoning about situation-specific preconditions, and is the way AP addressees the "qualification problem" [21].

A typical example of this type of reasoning in a robot application might go as follows. The robot is assigned the task of repairing a device. In the course of the planning process it might post a "protection interval" on the condition that a door remain open for the duration of some operation. Counterplanning

might discover that a human could close the door, and a way to prevent this could be to post a notice that the door must be open until further notice.

AP has other features which are needed to fully address the requirements of an intelligent agent. These include: reasoning about metric time (scheduling), execution monitoring, and replanning. Execution monitoring allows the agent using AP to recognize and skip plan steps that are overcome by events, and to replan when something unexpected occurs (e.g., a door that should have been left open is closed and locked). Execution monitoring also projects ground truth observations through the state space generated during planning. When ever there is a change in a output situation proposition, AP's execution monitor rechecks preconditions and recalculates certain "recomputable effects" of subsequent plan steps. This permits AP to predict failures, rather than waiting to notice them, as would happen if the agent relied only on situated reasoning and sequencing. This is obviously something the planner should be doing outside the sense-act cycle of the plan executor. In fact, AP was designed from the beginning to be a deliberative process aloof from the execution environment.

Replanning in AP is cued by the execution monitor when it notices or predicts failures. Replanning in AP is based on the concept of a "minimal repair wedge." AP assumes the majority of a plan is salvageable. The idea is to excise the minimum number of plan steps dependent on the failed step and replace them with a "wedge" of operations that achieve the original subgoal with alternate means. This strategy is both more computationally efficient and cognitively plausible than planning again from scratch, which is what most classical planners are doomed to do.

5 Implementation Status

We have completed implementation of an interface between a RAP-based sequencing system and a facility for providing a set of situated skills which the sequencer can manipulate to cause activity in the world [22, 25]. We are now combining AP with the RAP-based sequencer.

Shortly we expect to begin testing our architecture on a number of problems. Questions we plan to address include the following:

1. What activity is appropriate for planning or sequencing layers?
2. What domains where reactivity is sufficient?
3. Are there domains where sequencing is sufficient?
4. In which domains is deliberation highly useful?
5. To what measure is the architecture beneficial over more ad-hoc approaches?

6 Future Work

After we complete testing our architecture, we plan to explore the architecture as a basis to incorporate in robots certain cognitive capabilities normally associated with intelligent behavior. The first area we will investigate is learning.

We are guided in our ideas about learning by Anderson's ACT* model of cognition [2]. In ACT*, one of the important uses of learning is for performance improvement. It is hypothesized that expertise is gained by compiling common sequences of primitive actions into routines which henceforth take the agent essentially no time to derive. This type of learning can be easy in our implementation because the RAP library can be dynamically modified. For example, the robot could "learn" the commonly used plans for getting places from its nominal home (e.g., to the mess hall, the latrine). Since AP's plans are parameterized and have syntax similar to RAPs, the system could compress common sequences of operators into RAPs and then add that RAPs as a planning primitive. Thus, the next time the robot has a goal to eat dinner it would not have to invoke the planning system to accomplish the task of getting to the proper location.

References

[1] P. E. Agre and D. Chapman. Pengi - an implementation of a theory of activity. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 268–272, July 1987.

[2] J. R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[3] R. A. Brooks. Solving the find path problem by a good representation of free space. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 381–386, 1982.

[4] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[5] D. Chapman. Planning for conjunctive goals. Artificial Intelligence Laboratory 802, Massachusetts Institute of Technology, 1985.

[6] J. H. Connell. Creature design with the subsumption architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1987.

[7] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1992.

[8] J. G. de Lamadrid. Obstacle avoidance heuristic for three dimensional moving objects. Computer Science 87-16, Institute of Technology at the University of Minnesota, February 1987.

[9] C. Elsaesser and T. R. MacMillan. Representation and algorithms for multiagent adversarial planning. Department W153 MTR-91W000207, MITRE Corporation, December 1991.

[10] R. J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale University Department of Computer Science, January 1989. see Technical Report 672.

[11] R. J. Firby. Building symbolic primitives with continuous control routines. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems.* Morgan Kaufmann, June 1992.

[12] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* PhD thesis, Virginia Polytechnic Institute Department of Computer Science, April 1991.

[13] L. P. Kaelbling. Goals as parallel program specifications. In *Proceedings of American Association of Artificial Intelligence Conference*, 1988.

[14] L. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1993.

[15] J. P. Laumond. Model structuring and concept recognition: Two aspects of learning for a mobile robot. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 839–841, August 1983.

[16] T. Lozano-Perez and M. A. Wesley. An algorithm for planning collision free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[17] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, Massachusetts Institute of Technology, May 1990.

[18] D. P. Miller. Rover navigation through behavior modification. In *Proceedings of the NASA Conference on Spacecraft Operations Automation and Robotics*, July 1990.

[19] J. C. Sanborn and Hendler J. A. A model of reaction for planning in dynamic environments. *International Journal for Artificial Intelligence in Engineering*, 3(2):95–102, 1982.

[20] M. I. Shamos. *Computational Geometry.* PhD thesis, Yale University Department of Computer Science, 1975.

[21] Y. Shoham. *Reasoning about Change.* The MIT Press, 1988.

[22] M. G. Slack. Computation limited sonar-based local navigation. In *AAAI Spring Symposium on Selective Perception*, pages 142–146, April 1992.

[23] M. G. Slack. Sequencing formally defined reactions for robotic activity: Integrating RAPS and GAPPS. In *Proceedings of the SPIE Conference on Sensor Fusion*, November 1992.

[24] C. E. Thorpe. Path relaxation: Path planning for a mobile robot. In *Proceedings of American Association of Artificial Intelligence Conference*, pages 318–321, 1984.

[25] S. Yu, M. G. Slack, and D. P. Miller. A streamlined software environment for situated skills. In *Proceedings of the AAIA/NASA Conference on Intelligent Robots in Field, Factory, Service and Space*, March 1994.