Characterizing an Architecture for Intelligent, Reactive Agents

R. Peter Bonasso and David Kortenkamp
Metrica Inc. Robotics and Automation Group
NASA Johnson Space Center — ER4
Houston, TX 77058
bonasso or korten@aio.jsc.nasa.gov

Abstract

In this paper we briefly describe a software architecture that integrates reactivity and deliberation. The architecture has three levels, or tiers. The bottom tier contains a set of situated, reactive skills. The second tier is a sequencer that can activate sets of skills to perform a task. The third tier is a planner that reasons in-depth about goals, resources and constraints. We characterize this architecture across several dimensions and then describe the lessons that we have learned from applying this architecture to several robotics projects.

Introduction

For several years we have investigated ways to combine deliberation and reactivity in robot control architectures to program robots to carry out tasks robustly in field environments (Bonasso 1991; Bonasso, Antonisse, & Slack 1992). We believe this integration is crucial. Not only must an agent be able to adjust to changes in a dynamic situation, but it must be able to synthesize plans, since the complexities of the real world make precompiling plans for every situation impractical. We have arrived at an architecture that is an outgrowth of several lines of situated reasoning research in robot intelligence (Brooks 1986; Gat 1992). This architecture allows a robot, for example, to accept guidance from a human supervisor, plan a series of activities at various locations, move among the locations carrying out the activities, and simultaneously avoid danger and maintain nominal resource levels. We have used the architecture to program several mobile and manipulator robots in real world environments and believe that it offers a unifying paradigm for control of intelligent systems.

Our architecture separates the general robot intelligence problem into three interacting layers or tiers (and is thus known as 3T):

- A dynamically reprogrammable set of reactive skills coordinated by a skill manager (Yu, Slack, & Miller 1994).
- A sequencing capability that can activate and deactivate sets of skills to create networks that change the state of the world and accomplish specific tasks. For this we use the Reactive Action Packages (RAPs) system (Firby 1989).
- A deliberative planning capability that reasons in depth about goals, resources and timing constraints. For this we use a system known as the Adversarial Planner (AP) (Elsaesser & Slack 1994).

Figure 1 shows how these software layers interact. Imagine a repair robot charging in a docking bay on a space station. At the beginning of a typical day, there will be several routine maintenance tasks to perform on the outside of the station, such as retrieving broken items or inspecting power levels.

The human supervisor gives the robot a high-level goal – to conduct inspections and repairs at a number of sites – which the planner (deliberative layer) synthesizes into a partially-ordered list of operators. These operators would call for the robot to move from site to site conducting the appropriate repair or inspection at each site. For our example, we examine a subset of those which might apply to one site, namely 1) navigate to the camera-site-1 site, 2) attach to camera-site-1, 3) unload a repaired camera, 4) detach from camera-site-1. Each of these tasks corresponds to one or more sequenced set of actions, or RAPs (Firby 1989). The planner then enters a mode to begin executing its plan and monitoring the results. By matching via unification the planner's propositional effects clauses with the succeed clauses of the RAPs in the RAP library, the planner selects a navigate RAP to execute the first task.

The RAP interpreter (sequencing layer) decomposes the selected RAP\(^1\) into other RAPs and finally ac-
activates a specific set of skills in the skill level (reactive layer). Also activated are a set of event monitors which notifies the sequencing layer of the occurrence of certain world conditions. In this example, one of the events being monitored would be when the location of the end of the docking arm was within some specified tolerance of camera-site-1. When this event occurs, the clause \( (at \text{ robot } camera-site-I) \) would be posted to the RAP memory.

The activated skills will move the state of the world in a direction that should cause the desired events. The sequencing layer will terminate the actions, or replace them with new actions when the monitoring events are triggered, when a timeout occurs, or when a new message is received from the deliberative layer indicating a change of plan. In our example, the navigate RAP’s succeed clause \( (at \text{ robot } camera-site-I) \) would be true, terminating the RAP and causing the planner to label task one complete and to execute the attach task.

### Characterizing the architecture

There has been a great deal of work lately on integrating reaction and deliberation in a single architecture (Connell 1992; Musliner, Durfee, & Shin 1993; Schoppers 1987; Simmons 1990; Wilkins et al. 1995). While we do not have space for a direct comparison between our architecture and others, we can begin to characterize our architecture across several dimensions. This characterization is meant to show the motivations behind our architecture and demonstrate some design decisions that we have made along the way.

As stated above, our architecture has three levels of representation. All play a different role in coordinating the agent’s action. One important research issue is how to decide whether a certain aspect of a task belongs at the skill level, the sequencer level or the planning level. Our work in applying the architecture to the wide variety of projects (see Applications of the Architecture below) has led to a preliminary set of dimensions upon which to divide tasks across the levels.

The first dimension that we use for dividing a task is time. The skill level has a cycle time on the order of milliseconds; the sequencer level, tenths of seconds; and the planning level, seconds to tens of seconds. This imposes two constraints. First, if something must run in a tight loop (i.e., obstacle avoidance) then it should be a skill. Second, if something runs slowly (i.e., path planning) then it should not be a skill, as it will slow down the cycle time of the skill manager when it is active. Similar constraints hold when deciding whether something should be at the sequencer level or the planner level.

The second dimension that we use for dividing a task is bandwidth. The data connection between different skills in the skill manager is implemented using direct memory transfers. The data connection between the skill manager and RAPs is TCP/IP. Thus, we generally write skills that abstract perceptual information such that only small amounts of data are passed to RAPs. A RAP that requires a large amount of data (e.g., an image) should be written as a skill.

The third dimension that we use for dividing a task are the task requirements. Each level of the architecture has built-in functionality that makes certain operations easier. For example, RAPs has mechanisms for skill selection, so if a skill contains many methods for handling different contingencies, then it might be useful to break that skill into several smaller skills for each contingency and let a RAP choose among them. Similarly, if a RAP starts doing look-ahead search, resource allocation or agent selection, then it may be better off as a set of AP operators, which can then take advantage of AP’s built-in support for these functions.

The final dimension that we use for dividing a task is the modifiability that we desire. The skill manager is compiled on-board the robots. If we want to change a parameter or change the order of execution within a skill, we need to recompile the skill manager, which often means bringing down the entire robot system. RAPs and AP, on the other hand, are interpreted and connected to the robot system through TCP/IP. If we feel that a certain routine will require on-line modification by a human operator, then we want to put it at
the sequencer or planner level, not at the skill level.

Psychological motivations

Human cognition has been described in terms of levels of descriptions which are characterized by a particular time scale (Newell 1990). 3T lines up fairly well with Newell's characterization, i.e., a neural level at cycle times of less than 10 milliseconds (skills, reflexes), a cognitive level (our sequencer) at between 10 seconds and 10 milliseconds, and a rational band (the planner) at greater than 10 seconds.

But 3T also represents a smooth transition from event based reasoning to continuous action. This is somewhat different from earlier work (Chapman & Agre 1986), which posits that abstract reasoning grows out of our embodied actions. Moreover, 3T's transition attempts to model the human trait of attending to critical local, concrete activity, but being able to reason more abstractly when the activity becomes routine. An example is driving a car. On a long stretch of uninterrupted, flat, empty high speed highway, we will tend to look ahead to the next phase of the trip, assess whether to eat fast food or at a restaurant when we stop, or even plan a strategy to pursue when we get back to our work environment the following day. But as soon as the road begins to wind tightly, or fill up with other cars, we focus our attention on the proper sequencing of our sensing and driving skills to avoid an accident. All thoughts beyond the immediate second by second responses are abandoned.

Finally, 3T implicitly assumes there is always a plan of action. We believe there is support for this in human behavior, since most functional humans have a goal at any one point in time. This can be as far reaching as planning a career, or as mundane as satisfying basic needs such as food, safety, and survival. Further, it should be possible to use the 3T framework to investigate trade-offs between short and long term goals. There are many situations in life where humans mediate short term behavior for a more efficient achievement of long term gratification, such as when athletes in training restrict their diet and increase uncomfortable exercise in order to compete more effectively in a future race.

Learning

Although our architecture does not incorporate learning as yet, it was designed with learning in mind. We hypothesize three different ways of applying learning to our architecture.

First, learning can be applied within each layer of the architecture to increase that layer's performance and thus the performance of the architecture as a whole. There are many examples of learning being used to increase a planner's performance (e.g., (Knoblock, Minton, & Etzioni 1991)). The feedback that the RAPs system exploits from the low-level skills can be the basis of learning new methods in the sequencing layer, a process of learning by observation (e.g., (Dejong 1986)). There are also many examples of reinforcement learning being used to increase the performance of reactive robot skills, (e.g., (Brooks 1989)).

Second, learning can take place across layers, that is, activities that once required planning can, over time, be moved to the sequencer and finally to a skill. For example, when a plan is successful, the planner can create a new RAP that embodies the actions and variable bindings of that plan and add this RAP to the RAP library. Now the planner can use that new RAP as an atomic action within a larger plan. Similarly, if a RAP fails repeatedly, it is possible for the planner to change the context, preconditions or succeed clauses of that RAP in order to prevent future failures. Or, the planner could add a new context and method to an existing RAP in order to prevent failure during future executions.

Third, learning can be used to alter the response of the architecture to the environment. For example, the sequencer could learn the correct timings for each skill in a reactive package based on its experiences with the pace of the world around it. As the robot accumulates world knowledge it can better schedule its resources.

Interfaces

There are graphical user interfaces (GUIs) at each level of the architecture. At the planning level, a multiple pane GUI reminiscent of constraint-frames in LISP machines allows inspection of every aspect of a plan, as well as the input of goals via pop-up menus. The user may also control the planner's choices in the selection of subgoals, evaluation functions, and certainty factors.

At the sequencer level, domain specific point and click GUIs are developed to allow the user to invoke individual RAPs. This is particularly useful when a situation presents itself for which the planner cannot synthesize a response (due to insufficient operator or constraint knowledge), yet the human can devise an ad hoc method for success. This is equivalent to one agent "coaching" another through a new situation. It is at this level then, that a natural language interface on the order of Chapman's Sonya system (Chapman 1990) could be developed. Martin has developed a more comprehensive approach for natural language in the RAPs system (Martin & Firby 1991).

The skill level has a point and click GUI which allows the invocation of individual skills with a number of pa-
parameter settings, such as whether to record the I/O to a file for later playback, verbosity settings, and turning on and off any graphical display associated with a skill.

We have as yet not developed a set of interfaces which would allow the user to input new knowledge of plans (as in Cypress (Wilkins et al. 1995)) or of RAPs. This is partially due to focusing limited resources on the framework, and to our hope that certain machine learning techniques might obviate this need.

Applications of the architecture
This architecture has proven useful for the variety of projects described below. In some projects, we didn’t use the planner, only the skill manager and the sequencer. In all cases, we have found that using the architecture decreases the complexity of building and maintaining robot control systems.

Mobile robots for following and approaching
We have applied the skill manager and sequencer to three mobile robots projects that involve following and/or approaching. The first project is a mobile robot with a stereo vision system mounted on a pan/tilt/verge head and the task is to pursue people as they move around our laboratory. The skills on-board the robot are local navigation with obstacle avoidance (actually, four interacting skills) and visual acquisition and tracking of the target. The sequencer transitions the robot between acquisition, tracking and re-acquisition depending on the status of the visual tracking system. This robotic system successfully tracks people and other robots for up to twenty minutes (sometimes having to stop and autonomously re-acquire the target), while avoiding obstacles.

The second mobile robot project is a mobile base with a manipulator (see Figure 2). The task is to approach a workspace, use a wrist-mounted sonar sensor to find an aluminum can, grasp the can and carry it to another location. In this case, the skills on-board the robot are local navigation with obstacle avoidance, sonar-based docking with the workspace, sonar search for the object and manipulator movement and grasping. The sequencer transitions the robot between these skill sets, determining when each particular segment of the task is finished.

The third mobile robot project is a mobile base with a color vision system mounted on a pan/tilt head and the task is to locate and approach four different colors in a large room (see Figure 3). The skills on-board the robot are local navigation with obstacle avoidance and color searching, tracking and approaching. The sequencer determines a search strategy, activates different sets of skills at different phases of the search and reasons about false detections. This system repeatedly finds the four colors over an average twenty minute search period, even while being “tricked” with colors that appear momentarily and then disappear again, only to be seen later.
Manipulation tasks

We have applied the entire 3T architecture to a simulation of a three-armed EVA Helper/Retriever (EVAHR) robot carrying out maintenance tasks around a space station, much as described in the example used in the introduction. We then ported this system to a hardware manifestation of such a service robot in a dual-armed facility known as ARMSS (Automatic Robotic Maintenance of Space Station). In general, the difference between running 3T on a simulator and on actual robot hardware was primarily in the interfaces and the level of autonomy. The planner, RAPs, and the skill manager were essentially unchanged.

In the EVAHR simulation, once the plan is underway, users can interactively introduce failed grapple fixtures, failed arm joints, and gripper malfunctions. Simple failures such as failure of an arm or a grapple fixture are handled at the RAP level. Delays from these recoveries cause the planner to adjust the schedule of tasks at future sites. More drastic failures will cause the planner to abandon all tasks at a given site. And with enough malfunctions the planner abandons the entire plan and directs the robot back to its docking station.

At the ARMSS facility we wrote skill shells which invoked robot control software that already existed. In this manner, we were able to command the arms from RAPs within two weeks time. We also changed the locus of control from the planner to a human supervisor working at the RAPs level. This allowed the human supervisor to restart the skill layer and then re-invoke the current RAP whenever the hardware failed. This ability to allow a human supervisor to command lower level RAPs to extricate the robot in the case of a hardware problem was critical to completing several missions.

Lessons Learned

Our breadth of implementation across several projects has allowed us to gain insights into the strengths and weaknesses of the architecture. These insights are qualitative rather than quantitative. We believe that 3T can ease the development of software control code for most robot systems, which are notoriously complex. This is especially true in the case of multiple robotic subsystems. There are two reasons we believe this is true.

First, the skill manager framework abstracts away the need for the programmer to explicitly connect the data coming to and from a skill. This was especially evident in the mobile robot tracking project, where we used skills for movement and obstacle avoidance and a separate vision system with skills for tracking moving objects. When we integrated the two systems it was straightforward to feed the output of the vision tracking skill to the input of the obstacle avoidance skill so that the robot could follow people while still avoiding obstacles. Similarly, when we added a color tracking system to the same robot, the code integration was greatly simplified by the structure of the skill manager.

Second, as mentioned above, by decoupling the real-time execution of skills from sequencing and planning the use of those skills, we allow for modifications of sequences and plans without having to reinitialize the robot controllers. Our approach lends itself naturally to a bottom-up approach to programming robots whereby lower level skills are written and debugged separately, before being integrated together to accomplish a task.

Future work and conclusions

We have described a robot control architecture that integrates deliberative and situated reasoning in a representational framework that flows seamlessly from plan operators to continuous control loops. The architecture has been demonstrated successfully in a wide range of mobile and manipulator robot projects, both real and simulated. We have found that the division of labor among the layers of the architecture permits the generalization of knowledge across multiple projects. We have also found that our software tools allow for rapid implementation of complex control systems.

We are also exploring the use of 3T for closed ecological support systems (CELSS). Previous CELSS experiments such as those conducted in the U.S. and in Russia have shown that most of the crew's time is spent in crop management and monitoring environmental control systems. In an effort to automate some of these processes we have developed the skill and sequencing layers of the architecture to control a simulation of an o2-co2 gas exchange system with a crew of three and a crop of wheat. The skills consist of setting valve openings, plant lighting levels, suggesting crew activity and monitoring the gas flows and the storage levels. We are also developing AP plan operators which will determine the planting cycles of various crops to support gas exchange as well as dietary requirements of the crew.

Having achieved this framework we have begun to investigate the integration of other AI disciplines. Natural language is already being researched at the RAPs level (Martin & Firby 1991). And the architecture could benefit from combining with concurrent perception architectures such as those supporting mapping (Kuipers & Byun 1991; Kortenkamp & Weymouth, 1991).
References


