Experiences with an Architecture for Intelligent, Reactive Agents

R. Peter Bonasso[1], David Kortenkamp[1], David P. Miller[2] and Marc Slack[2]

[1] Metrica, Inc. Robotics and Automation Group
NASA Johnson Space Center — ER4
Houston, TX 77058
bonasso@aio.jsc.nasa.gov

[2] The MITRE Corporation – MS Z421
7525 Colshire Drive
Mclean, VA 22102

Abstract. This paper briefly describes a robot architecture that has been under development for the last eight years. This architecture uses several levels of abstraction and description languages that are compatible between levels. The makeup of the architecture helps to coordinate planful activities with real-time behaviors for dealing with dynamic environments. In recent years, many architectures have been created with similar attributes. The two features that distinguish this architecture from most of those are: 1) a variety of useful software tools have been created to help implement this architecture on multiple real robots; and 2) this architecture, or parts of it have been implemented on over half a dozen very different robot systems using a variety of processors, operating systems, effectors and sensor suites.

1 Introduction

Since the late eighties we have investigated ways to combine deliberation and reactivity in robot control architectures to program robots to carry out tasks robustly in field environments [18, 19, 24, 20, 1, 22]. 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 [3, 11, 8, 10, 4, 27]. 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[32].
- 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 [7].
- 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) [6].

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 that 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; and 4) detach from camera-site-1. Each of these tasks corresponds to one or more sequenced set of actions, or RAPs [7]. The planner then enters a mode to begin executing its plan and monitoring the results. By matching via unification the planner's propositional purpose 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[3] into other RAPs and finally activates a specific set of skills in the skill level (reactive layer). Also activated are a set of event monitors that 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 aligned within some specified tolerance with camera-site-1. When this event occurs, the clause *(at robot camera-site-1)* 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 robot camera-site-1)* would be true, terminating the RAP and causing the planner to label its navigate task complete and to execute the attach task.

[3] A given RAP can represent a complex, though routine, procedure for accomplishing a task. For instance, in one of our manipulator projects, unloading an item involves unfastening bolts, two or three tool changes, and the use of a redundant joint capability.

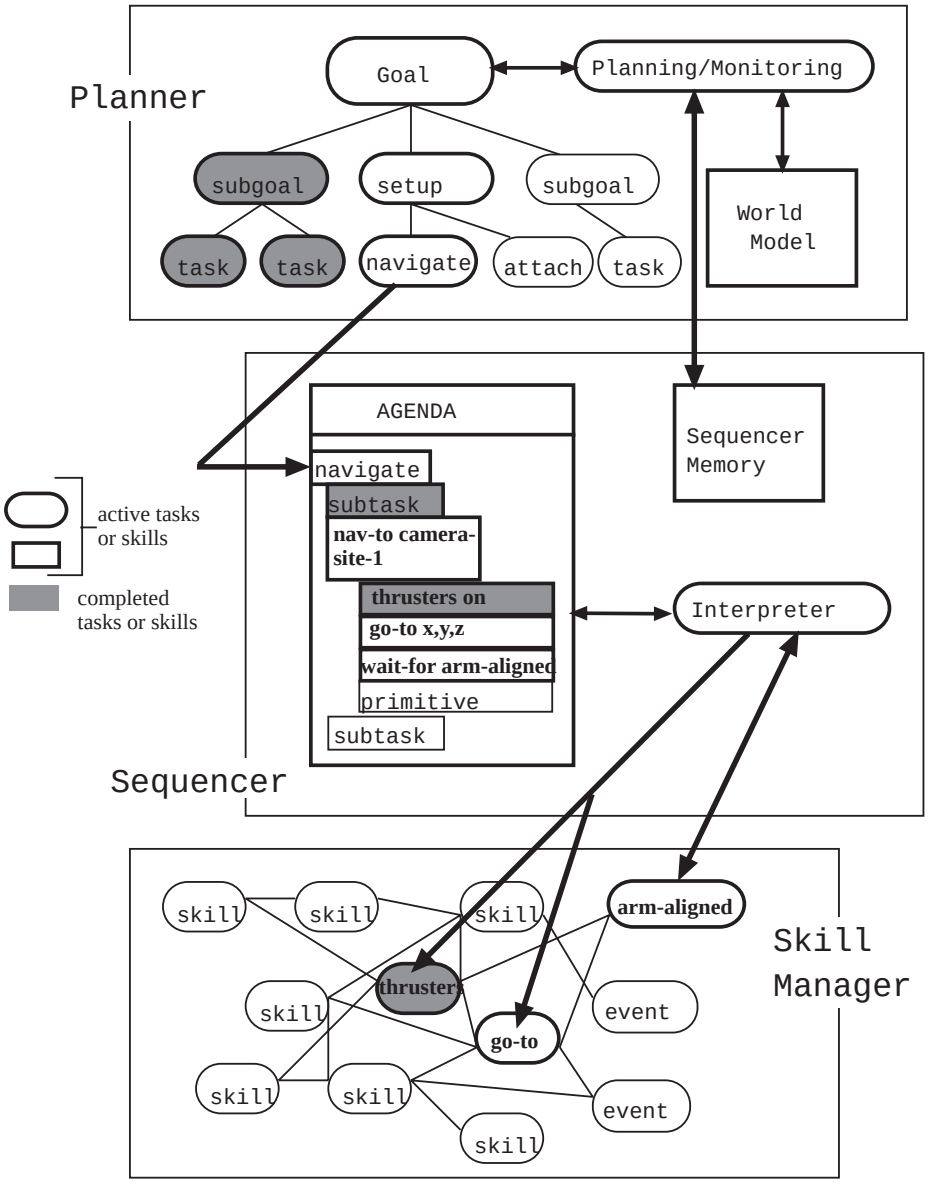

Fig. 1. 3T Intelligent Control Architecture

2 Software Tools for Architecture Implementation

To support these ideas, and to make the architecture explicit, we have developed a number of tools and systems for integrating these three layers and for providing the user with a paradigm for developing robotic applications within the architecture.

2.1 Skills

We have created a program called a skill manager which acts as the interface between a set of situated skills and the rest of the architecture. Situated skills represent the architecture's connection with the world. The term situated skills [28] is intended to denote a capability that, if placed in the proper context, will achieve or maintain a particular state in the world. For example, one might develop a situated skill for grasping a handle. Such a skill will only be useful if the robot is currently located in front of a handle; in other situations the skill will fail. Skills form the robot-specific interface with the world, handling the real-time transformation of desired state into continuous control of the motors and interpretation of the sensors.

As robots and other systems can vary greatly in their physical characteristics and sensor capabilities, the skills that make up the robot's interface with the world also vary greatly between robots and environments. However, because we would like a control architecture that is robot independent, we have developed a set of tools for constructing these situated skills [32]. This canonical approach to skill development forces a standard interface among the skills and a standard interface to the sequencer, independent of the physical characteristics of the robot and sensors to which they are connected. The representation includes:

1. The skill's input and output specification. Each skill must provide a description of the inputs it expects and a listing of the outputs that it generates. This allows skills to be networked by having the outputs of one skill automatically routed to the inputs of another skill.
2. A computational transform. This is where the skill does its work. This function will be provided with inputs and must generate outputs. This computation may be run synchronously (i.e., computed each time the function is called) or asynchronously (returns immediately with the latest results).
3. An initialization routine. Each skill is given the opportunity to initialize itself when the system is started (e.g., setup communications ports, etc.).
4. An enable function. The sequencer can enable and disable skills. Depending on the context, a skill is given the opportunity to perform any special start up procedures each time it is enabled after having been disabled.
5. A disable function. When a skill is no longer needed, the sequencer can disable it. The disable function performs any necessary cleanup actions.

From the sequencer's perspective, skills must be capable of being enabled and disabled in any combination depending on the situation. Except for common input and output data structures, each skill is totally independent of the

others. At any one time the reactive layer of the architecture is characterized by the currently enabled network of skills. To provide the sequencer with a uniform interface and to allow the skills to communicate with each other, the skill development environment encapsulates the skills inside the skill manager. The skill manager also handles the interface with the sequencing system by providing all of the communications and asynchronous events that the sequencer needs in order to stay coordinated with the skills.

Through the ability to reconfigure the system depending on the aspect of the task at hand, the architecture allows developers to focus effort on the important facets of particular tasks without having to be overly concerned with the way in which the skills as a whole interact to generate coherent task-directed behavior. The determination of how best to configure the skills for the situation at hand is the task of the sequencing layer.

2.2 Sequencing

To accomplish tasks that the robot must routinely perform, the architecture has a sequencing system. In our case, the sequencer is the RAPs interpreter. In its simplest form, a RAP is a description of how to accomplish tasks in the world. For example, a RAP for docking with the hull of a ship might have the following form:

```
(define-rap (attach-at-site ?thesite)
  (succeed (docked ?theship))
  (method m1
    (context (ferrous-hull ?thesite))
      (sequence
        (t1 (approach-site ?thesite) (for t2))
        (t2 (magnetically-attach ?thesite)
            (wait-for (docked ?thesite)))))
  (method m2
    (context (not (ferrous-hull ?thesite)))
      (sequence
        (t1 (approach-site ?thesite) (for t2))
        (t2 (grip-attach ?thesite)
            (wait-for (docked ?thesite))))))
```

Notice that the way the task is accomplished is dependent upon the robot's knowledge of the situation. So, in the above example the robot accomplishes the task of *attach-at-site* differently depending on whether the spacecraft's hull is ferrous or not. Further distinctions could be made depending on the size of the craft or on any other task relevant feature. The sequencer contains a library of such RAPs, each keyed to specific situations and each activating a different set of skills in order to accomplish its particular task. In the above example, the `wait-for` statements cause the RAP interpreter to block that branch of the task execution until a reply is received from the skill manager. Replies are produced by special skills called *events*. Events take inputs from other skills and notify the

sequencer whenever a desired state has been detected. Thus, events represent the mechanism by which the sequencer determines when a particular subtask has been accomplished as well as keeps track of changes in the state of the world.

The RAP interpreter runs in a continuous loop installing new goals on its agenda, deinstalling old goals that have been achieved or that have failed, decomposing higher level RAPs into lower level RAPs, activating or deactivating skills and watching for responses from event skills. On each cycle of the interpreter, for each skill that needs activation or deactivation, an activate or deactivate message is sent to the skill manager which acknowledges the messages and activates or deactivates the skills. Usually the sequencer will have requested activation of event skills; and in those cases on each cycle it listens (on the port or communications socket) for a response from the skill manager that an event has occurred. So in the example above, for method m1, the `approach-site`,`magnetically-attach` and `docked` skills will be activated. Assuming there are no other RAPs pending, the interpreter will just cycle listening for an event-occurred message from the skill manager. When the `docked` event occurs, such a message will be sent by the skill manager, and when the sequencer receives it, an event memory rule such as

```
(define-memory-rule (attach-at-site ?thesite) :event
  (rule
    (t
      (mem-add (docked ?theship)))))
```

will fire, adding the `(docked ?theship)` proposition to the RAP memory. Since this will unify with the succeed clause, the RAP will terminate successfully.

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. Nor are these two layers efficient 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., unload a camera, move to a site), it lacks the ability to string these routine tasks together in order to manifest the required "global" behavior. The ability to consider the global implications of actions is exactly the task for which deliberative planners are designed.

2.3 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 difficult to specify as sequences of common robotic skills. When planning is necessary, the planner operates at the highest level of abstraction possible so as to make its problem space as small as possible.

The role of reaction is to control real-time behavior. The role of sequencing is to control rote or iterative series of real-time behaviors. In the process, the sequencing layer will raise the level of abstraction of the activities with which the planner will concern itself. This simplifies the planning problem because it

lets a few operators stand for many essentially linear planning problems. The 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. Importantly, all three layers must operate concurrently and asynchronously. Accomplishing this is the key to making planning useful in a robot.

The planner we are using in our experiments, AP [5], is similar to many state-based, hierarchical, non-linear planners. It searches a space of plans to decompose a highly abstract goal into more detailed subgoals, using preconditions and effects as the constraints. AP also has a number of features which make it compelling to use for robot planning. One aspect of intelligent robots overlooked by both the planning and robot control communities is that robots will normally not be fully autonomous but will be working in conjunction with other agents – a human giving orders as a minimum. Multiagent control is necessary when more than one robot is employed to carry out tasks, when a single robot has to coordinate the use of its own resources (e.g., arms and grippers), or when multiple robots are operating independently on multiple tasks in a shared environment.

AP was designed to deal with multiagent coordination by extending its 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 any agent 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. This particular temporal constraint also affects the temporal duration calculations AP uses when it constructs a schedule to go with its plan, and AP's plan execution monitor. Certain other temporal constraints in AP's language allow (but do not mandate) codesignation.

```
(Operator grasp-bulky-object
  :purpose (holding ?planner ?large-thing)
  :arguments
    ((?size-of-thing
       (get-value ?large-thing 'size)))
  :preconditions
    ((top ?large-thing clear)
     (on ?large-thing ?something))
  :constraints
    ((can-lift ?arm-or-robot1
               (* 0.5 ?size-of-thing))
     (can-lift ?arm-or-robot2
              (* 0.5 ?size-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 attractive feature of AP for this work is that it can reason about uncontrolled agents – a result of its original development for adversarial planning. An uncontrolled agent might be a human operating in the environment along with a robot, or even nature. AP uses a `counterplanning` mode to reason about how preconditions or during conditions in a plan might be negated by an uncontrolled agent, thus thwarting the plan. These problems are addressed by augmenting the plan with operations that prevent the negative effects of the uncontrolled action. This amounts to reasoning about situation-specific preconditions, and is the way AP addresses the `qualification problem` [25]. AP can use these adversarial reasoning capabilities as a risk assessment mechanism to consider the probability of dangerous interactions with other agents.

The interface of the planner to the RAPs system takes place during execution monitoring and is tied to the primitive planner operators – i.e., those operators without plots. In general, AP's execution monitor examines the current plan in light of the current perceived situation, and decides a) which agents have plan steps that can be executed, and b) which next step each agent should carry out. In the most basic interface, the execution monitor unifies the purpose of each agent's next activity with a clause in the succeed clause of a RAP. For example, a purpose clause of a `go-to-place` operator might be `(at ?planner ?place)` which might be one of the propositions in the succeed clause in a "take-up-position-at" RAP. Having found a RAP with such a succeed clause for each agent, the execution monitor installs the instantiated RAPs as goals on the RAP agenda. If no RAP has such a clause, an error condition exists (this would be discovered when the primitive operator is defined). When the RAP system has no more goals to execute, the execution monitor updates its perceived situation *by querying the RAPs memory* and then generates the next activity list.

To support replanning, the AP execution monitor also checks on agents, time, and preconditions. The status of agents is made available from the RAP memory and can trigger the reassignment of agents for future tasks. The current time is maintained in RAP memory, and AP uses it to judge the progress of the plan and to adjust the plan schedule of earliest and latest start and end times for each operator as well as the final completion time of the plan itself. Finally, the preconditions used in the primitive plan operators are ascertained by the execution monitor from the RAP memory in the same manner as the purpose clauses.

2.4 Approaches to achieving coherent overall behavior

There are at least three different types of planner control that we have structured 3T to support: top down, problem-solving, and concurrent planning and execution. Top down control of multi-tiered architectures is the traditional structure for AI planning systems. Goals come in to the planner, the planner creates

a plan to accomplish the goals, and the plan is executed one step at a time. The 3T architecture supports this technique by allowing a planner to hand plan steps (or a whole plan) to the sequencer via the task agenda. The planner can then monitor plan execution by waiting for sequencer messages that signify the completion of each task.

An alternative flow of control focuses on the sequencer. Methods can be included at the sequencing level to recognize when certain situations arise and then call special purpose problem solvers to deal with those situations as needed. For example, a sequencer method for moving from one place to another might call a route planner to construct an appropriate set of intermediate steps, or a method that needs to mate two parts might call the planner to figure out the best manipulator moves to make.

The 3T architecture is also designed to support a very dynamic flow of control, with the planner and sequencer running concurrently and each making changes to the task agenda. The sequencer takes tasks to execute from the agenda and adds new subtasks generated during execution. The planner adds and deletes tasks from the agenda, as well as placing ordering constraints on tasks already in the agenda. Thus, the planner tries to arrange tasks to create a better future for the robot even while the sequencer is refining and executing those tasks. This concurrent flow of control is discussed in detail in [12] but has not yet been implemented.

Using any, or all, of these models of control flow within 3T, the goal is to support a tight integration of deliberative and reactive control. The planning tier (and problem solvers) use the sequencer's routine procedures as primitive operators. Since the sequencer can deal with many of the details of a situation itself, plans can be synthesized at a higher level of abstraction than would otherwise be possible, resulting in faster plan generation and replanning. Further, plans need only be partially ordered to accommodate any constraints at the planner's level of abstraction; the sequencer can instantiate the actual order at execution time. The only real restriction is that a top-down planner must be cognizant of the success or failure of the routine procedures of the sequencer, so that appropriate replanning can be invoked when appropriate.

3 Applications of the architecture

This architecture, and its associated tools have proven useful for the variety of projects described below. Our experience with 3T has shown that it is imminently practical to implement. RAPs and plan operators generalize across projects with mobile robots and among those with manipulator robots. This makes the porting of a full deliberative/reactive capability among those projects essentially one of writing the low-level skills. Moreover, if a body of low-level software exists to begin with, developing the skill level reduces to writing an appropriate I/O wrapper around the existing functions so as to allow them to be enabled by the skill manager. In many projects, we didn't use the planner, only the skill manager

and the sequencer. In this section, we present the simplest implementation first and then increasingly more complete implementations.

3.1 Robotic wheelchair

We have used the skill manager and an abstraction of the sequencing layer to control a robotic wheelchair[21]. This application uses a computer-controlled wheelchair outfitted with a variety of proximity and range sensors. The groups of skills are activated by pressing a particular button. This allows the user to turn on hall following skills (for example) at the touch of a single button, greatly reducing the required interaction normally needed for a person to guide an electric wheelchair.

3.2 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 switches the robot skill set configuration between acquisition, tracking and reacquisition depending on the status of the visual tracking system. This robotic system has successfully tracked people and other robots for up to twenty minutes (sometimes having to stop and autonomously reacquire the target), while avoiding obstacles [13].

The second mobile robot project is a mobile base with a manipulator. 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 essentially reconfigures the robot's skill set, 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. 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 [31].

3.3 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, two people 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 reinvoke 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.

3.4 Mobile robot errand running

We have applied the entire architecture to one mobile robot project, a Denning robot whose task is to run simple errands. The robot contains skills that allow it to follow walls, go to markers and to avoid obstacles and has a RAP library with sequences for going from place to place, going through doors, checking on the status of elevators, looking for landmarks, etc. With these routines on-board, we are able to give the robot goals such as "check if the service elevator is available." The robot would then use AP to construct a plan to find its present location, plan a path to the elevator, navigate out the room, through the door, down the hall, and to the elevator. Because the planner was used, if the robot's path is blocked, it can replan its way. It evaluates the revised plan to make sure that no deadlines are violated. If the path to the elevator is blocked, and the resulting go-around is too lengthy, the robot will immediately abandon that goal and return and report failure.

4 Allocating knowledge across the architecture

Our core software tools, along with many of the RAPs and AP operators are easily transportable across our projects. The individual skills and events are easily transportable across different projects using the same platform, but tend to be hardware specific. 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 described above 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.

5 Evaluating and comparing the architecture

Our breadth of implementation across several projects has allowed us gain insights into the strengths and weaknesses of the architecture. These insights are qualitative rather than quantitative. However, there were large quantitative differences in development time for getting our robots to change tasks within the same domain. These tasks just involved changes at the RAP or AP operator level and no changes at the skill level, as compared to changing the behavior of a Subsumption [3] controlled robot where the changes percolated through the entire robot's code.

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.

Of the robot architectures in use, 3T has its strongest similarity to ATLANTIS [10] since they both grew out of the same research history. ATLANTIS embodies the "sequencer in control" approach to coherent behavior, whereas most of the examples used in this paper espouse the "planner in control approach". In ATLANTIS, the deliberative layer must be specifically called by the sequencing layer when the sequencer has nothing else to do. Additionally, ATLANTIS' skill level follows a strict data flow model. As such, skill events which latch certain conditions, are not explicitly supported (though much of their functionality can be duplicated by the use of state variables). The latching capability of the events in 3T is such that there are no real-time requirements on the sequencing layer.

3T shares many aspects of Cypress [30]. Our AP planner is comparable in representational power to SIPE; RAPs compares favorably with PRS. But because RAPs were designed to allow integration with conventional AI planners, we did not have to write an interlingua such as ACTs to achieve such integration. Additionally, Cypress does not include a canonical interface to different robot controllers as does 3T's skill manager.

TCA [26] has been used for a number of robot projects. TCA consists of a set of task-specific computational processes which communicate with each other by passing messages through a central control module. TCA is essentially a specialized operating system for physical devices (i.e., robots). While a process can be a robot behavior or a task planner, there is no real notion of layers, or of integrating plan synthesis and reactive execution as in ATLANTIS, Cypress, and 3T. The MIX multi-agent architecture [14] is very similar to TCA, but again, it is a flat network structure for agents, not a layered framework for integrating deliberation and reactivity.

While CIRCA [23] has been used mostly with simulations, it does make a

strong claim for meeting hard-real time constraints. While we have the ability in our architecture to request that specific skills run at a certain frequency, we have no way of enforcing this request as there is no interruption of skills. As our robots take on more complex tasks, we believe that we will need to address hard real-time issues.

Several papers in this volume report on the development of architectures that appear on the surface to be similar to 3T. The INTERRAP model [9] also uses three layers: a behavior layer, a local planning layer and a cooperative planning layer. The behavior layer corresponds to the sequencing layer in 3T and the local planning layer corresponds to 3T's deliberative layer. The authors chose to "layer" additional planning specifically for multi-agent activity atop the other layers. 3T has addressed only top-down control of multiple agents, i.e., a single "coordinator" at the planning level, not cooperative schemes. But INTERRAP underestimates the importance of a principled mechanism (the sequencing layer) for transforming the continuous activity of the robot into the discrete events needed by the higher levels of the architecture. As the INTERRAP research moves away from simulations to deal with physical robots with advanced perception capability, we believe this lack will become more apparent. It seems that with only a small amount of effort the INTERRAP behavior layer could be modified to activate and deactivate skills via a skill manager as in 3T.

The SIM_AGENT work [29] shares a similar lack. This tool kit for constructing agents blurs the distinction between continuous physical activity and symbolic state changes. Some operations in the SIM_AGENT framework are "automatic" and by-pass "normal" processing. Others require some deliberation. But all use fairly traditional rules and object-oriented methods and procedures. This makes for easy construction of a variety of approaches to agent operations, but will most likely require something like the 3T skill manager capability to be effective with physical robots.

6 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. The architecture comes with a set of software tools which allow the rapid implementation of complex control systems.

The modularity of the architecture provides several benefits. A task-achieving robot can be realized with just the first or the first and second tiers, albeit at a reduced level of functionality. Also, in our experience, while the skills tend to be robot-specific, the RAPs and plan operators generalize at least among manipulators and among mobile platforms. As well, the bottom-up development allows one to add functionality incrementally to the robot, as an increased skill base gives rise to more task-level capability, which in turn provides for more flexibility in plan development.

Probably the most fundamental result of this work is a clear demonstration of the power of abstraction from the continuous activity of the skills level, to the propositional representation of the sequencer, to the more abstract propositions of the planning level. The sequencer abstracts present continuous activity into discrete states, and discrete states into more abstract states. The planner transforms those abstract states into more abstract states which are useful for inferring the future implications of the sequencer's past activities. With the reduced search spaces, conventional planning and replanning techniques take on more power, and the robot can be made to exhibit a level of rationality not possible with other approaches.

Recently, we have begun to investigate 3T's use in non-robotic control systems. One example is a modified 3T that acts as a World Wide Web (WWW) robot. This is being accomplished by augmenting the skill layer of the architecture with a set of primitives for retrieving and manipulating Universal Resource Links (URLs). The task description language of the AP and RAP systems lends itself to the kinds of activities taken when users must respond to environmental disasters. For example, in response to a forest fire, the WWW robot searches the Web to retrieve maps of the location, and then use those maps to create a logistics plan for fighting the fire. This work is just beginning, but the task and planning languages embodied in the architecture lend themselves neatly to the creation of interactive decision aids which require both "sensing" and "action".

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 [17]. Machine learning techniques can be investigated from case-based reasoning in the planning layer to reinforcement learning in the skill layer [2]. And the architecture could benefit from combining with concurrent perception architectures such as those supporting mapping [16, 15].

References

1. R. Peter Bonasso, H.J. Antonisse, and Marc G. Slack. A reactive robot system for find and fetch tasks in an outdoor environment. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.
2. R. Peter Bonasso and David Kortenkamp. An intelligent agent architecture in

which to pursue robot learning. In *Proceedings of the MLC-COLT '94 Robot Learning Workshop*, 1994.

3. Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.
4. Jonathon H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings IEEE International Conference on Robotics and Automation*, 1992.
5. Chris Elsaesser and Richard MacMillan. Representation and algorithms for multiagent adversarial planning. Technical Report MTR-91W000207, The MITRE Corporation, 1991.
6. Chris Elsaesser and Marc G. Slack. Integrating deliberative planning in a robot architecture. In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*, 1994.
7. R. James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.
8. R. James Firby. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems*, 1994.
9. Klaus Fischer, Jörg P. Müller, and Markus Pischel. Unifying control in a layered agent architecture. In *Proceedings of the IJCAI 1995 Workshop on Agent Theories, Architectures, and Languages*, 1995.
10. Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1992.
11. Erann Gat, R. James Firby, and David P. Miller. Planning for execution monitoring on a planetary rover. In *Proceedings of the Space Operations Automation and Robotics Workshop*, 1989.
12. Steve Hanks and R. James Firby. Issues and architectures for planning and execution. In *Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 59–70, San Diego, CA, November 1990. DARPA.
13. Eric Huber and David Kortenkamp. Using stereo vision to pursue moving agents with a mobile robot. In *1995 IEEE International Conference on Robotics and Automation*, 1995.
14. Carlos A. Iglesias, José C. Gonzalez, and Juan R. Velasco. MIX: A general purpose multiagent architecture. In *Proceedings of the IJCAI 1995 Workshop on Agent Theories, Architectures, and Languages*, 1995.
15. David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.
16. Benjamin J. Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8, 1991.
17. Charles E. Martin and R. James Firby. Generating natural language expectations from a reactive execution system. In *Proceedings of the 13th Cognitive Science Conference*, Chicago, IL, 1991.
18. David P. Miller. A plan language for dealing with the physical world. In *Proceedings of the Third Annual Computer Science Symposium on Knowledge Based Systems*, Columbia SC, 1986.
19. David P. Miller. Execution monitoring for a mobile robot system. In *Proceedings of the 1989 SPIE Conference on Intelligent Control and Adaptive Systems*, pages 36–43, Philadelphia PA, 1989.

20. David P. Miller. Autonomous rough terrain navigation: Lessons learned. In *Proceedings of Computing in Aerospace 8*, pages 748–753, 1991.
21. David P. Miller and Marc G. Slack. Increasing access with a low-cost robotic wheelchair. In *Proceedings of IROS '94*, pages 1663–1667, 1994.
22. David P. Miller, Marc G. Slack, and Chris Elsaesser. An implemented intelligent agent architecture for autonomous submersibles. In *Intelligent Ships Symposium Proceedings: Intelligent Ship Technologies for the 21st Century*, 1994.
23. D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, and C. J. Paul. The challenges of real-time AI. *IEEE Computer*, 28(1), 1995.
24. Jim Sanborn, B. Bloom, and D. McGrath. A situated reasoning architecture for space-based repair and replace tasks. In *Goddard Conference on Space Applications of Artificial Intelligence (NASA Publication 3033)*, 1989.
25. Y. Shoham. *Reasoning about Change*. MIT Press, Cambridge, MA, 1988.
26. Reid Simmons. An architecture for coordinating planning, sensing and action. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.
27. Marc G. Slack. Computation limited sonar-based local navigation. In *Proceedings of the AAAI 92 Spring Symposium on Selective Perception*, 1992.
28. Marc G. Slack. Sequencing formally defined reactions for robotic activity: Integrating raps and gapps. In *Proceedings of SPIE's Conference on Sensor Fusion*, 1992.
29. Aaron Sloman and Riccardo Poli. SIM_AGENT: A tookit for exploring agent designs. In *Proceedings of the IJCAI 1995 Workshop on Agent Theories, Architectures, and Languages*, 1995.
30. David E. Wilkins, Karen L. Myers, John D. Lowrance, and Leonard P. Wesley. Planning and reacting in uncertain dynamic environments. *Journal of Experimental an Theoretical AI*, 7, 1995.
31. Carol Wong, David Kortenkamp, and Mark Speich. A mobile robot that recognizes people. In *Proceedings of the 1995 IEEE International Conference on Tools with Artificial Intelligence*, 1995.
32. Sophia T. Yu, Marc G. Slack, and David P. Miller. A streamlined software environment for situated skills. In *Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94)*, 1994.

This article was processed using the LaTeX macro package with LLNCS style