Experiences with an Architecture for Intelligent, Reactive Agents

R. Peter Bonasso[*], R. James Firby[†], Erann Gat[‡],
David Kortenkamp[*], David P. Miller[§], Marc G. Slack[§]

[*]Metrica Incorporated
Robotics and Automation Group
NASA Johnson Space Center – ER4
Houston, TX 77598
{bonasso,korten}@mickey.jsc.nasa.gov

[†]Computer Science Department
University of Chicago
1100 East 58th Street
Chicago, IL 60637

[‡]Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109

[§]The MITRE Corporation
7525 Colshire Dr.
McLean VA, 22102

Abstract

This paper describes an implementation of the 3T robot architecture which has been under development for the last eight years. The architecture uses three levels of abstraction and description languages which 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, other architectures have been created with

similar attributes but two features distinguish the 3T architecture: 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 a variety of very different robot systems using different processors, operating systems, effectors and sensor suites.

1 Introduction

Since the late eighties we have been investigating ways to combine deliberation and reactivity in control architectures for programming robots to carry out tasks robustly in field environments [Miller, 1986; Firby, 1987; Gat *et al.*, 1989; Bonasso *et al.*, 1992; Kortenkamp *et al.*, 1993; Miller *et al.*, 1994]. We believe this integration is crucial. Not only must an agent be able to adjust to changes in a dynamic situation, it must also be able to synthesize plans; 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; Firby, 1989; Gat, 1992; Connell, 1992; Slack, 1992a]. The architecture allows a robot, for example, to plan a series of activities at various locations, move among the locations carrying out the activities, and simultaneously avoid danger, maintain nominal resource levels, and accept guidance from a human supervisor. 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. The particular implementation of 3T described in this paper consists of:

- A dynamically reprogrammable set of reactive skills coordinated by a skill manager[Yu *et al.*, 1994].
- A sequencer that activates and deactivates 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 planner that reasons in depth about goals, resources and timing constraints. For this we use a system known as the Adversarial Planner (AP) [Elsaesser and Slack, 1994].

Figure 1 shows how these software tiers 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. In addition, a human supervisor assigns the robot a set of inspection and repair tasks at a number of sites around the station.

The planner (deliberative tier) synthesizes all of these goals into a partially-ordered plan listing tasks for the robot to perform. These tasks would call for the robot to move from site to site conducting the appropriate repair or inspection at

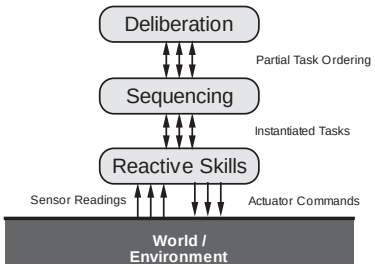

Figure 1: The 3T Intelligent Control Architecture

each site. For our example, we examine a subset of those tasks 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 sets of sequenced actions, or RAPs [Firby, 1989]. The planner then begins executing its plan and monitoring the results. By matching via unification each task's propositional effects clauses with the succeed clauses of RAPs in the RAP library, the planner selects a navigate RAP to execute the first task.

The RAP interpreter (sequencing tier) decomposes the selected RAP[1] into other RAPs and finally activates a specific set of skills in the skill level (reactive tier). Also activated are a set of event monitors which notifies the sequencing tier 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 fact *(at robot camera-site-1)* would be posted to the RAP memory.

The activated skills (reactive tier) will move the state of the world in a direction that should cause the desired events. The sequencing tier 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 tier 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 task one complete and move on to execute the next task.

2 Software Tools for Architecture Implementation

To support the use of these architectural ideas, we have developed a number of tools and systems for integrating the three tiers together and providing the user with a paradigm for developing robotic applications.

[1] 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.

2.1 Skills

Situated skills represent the architecture's connection with the world. The term *situated skill* [Slack, 1992b] is intended to denote a configuration of the robot's control system that, if placed in the proper context, will achieve or maintain a particular state in the world. In practice, control system configurations are created by enabling a set of skills that work together in a given context. For example, one might develop a situated skill for grasping a handle that uses a skill to visually track the handle, a skill to move the hand toward the target, and a skill to close the gripper on contact. Such a collection of skills will be useful if the robot is currently located in front of a handle but might fail in other situations. 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. So, to keep the control architecture robot independent, we have developed a set of tools for constructing these situated skills [Yu *et al.*, 1994]. 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. Once a skill is enabled, it uses this transform to continually recompute its outputs based on its current inputs.

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.

5. A disable function. When a skill is no longer needed, the sequencer will disable it and 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 tier 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.

The ability to reconfigure the system according to various aspects of the task at hand allows the developer 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 tier.

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 simply a description of how to accomplish a task in the world under a variety of circumstances using discrete steps. For example, a RAP for docking with the hull of a spaceship might have the following form:

```
(define-rap (attach-at-site ?thesite)
  (succeed (docked ?thesite))
  (method
    (context (ferrous-hull ?thesite))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (t2 (magnetically-attach ?thesite)
            (wait-for (docked ?thesite))))))
  (method
    (context (not (ferrous-hull ?thesite)))
    (task-net
      (sequence
        (t1 (approach-site ?thesite))
        (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 spaceship'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, the sequencer uses events to determine when a particular set of skills has completed its work and when particular states of the world have changed.

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 tiers is not structured to perform complicated resource allocation reasoning. Nor are these two tiers efficient at reasoning about the failure requirements or consequences of a task. So while the sequencer has the ability to handle routine situations (e.g., unload a camera, move to a site), it lacks the foresight to organize novel sequences of routine tasks to manifest a required "global" behavior. The ability to consider the global implications of actions is 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 not have to deal with tasks that can be routinely specified 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 generate well-known series of real-time behaviors. In the process, the sequencing tier 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 large families of similar execution time actions. The ability of the RAP system to deal with iterative behavior greatly simplifies the planner's representation, allowing a propositional state representation common to classical planning to suffice in many common situations. Importantly, all three tiers 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 [Elsaesser and MacMillan, 1991], 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.

```
(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 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 addressees the `qualification problem` [Shoham, 1988]. AP can use these adversarial reasoning capabilities as a risk assessment mechanism to consider the probability of dangerous interactions with other agents.

3 Applications of the Architecture

We have applied our architecture to several robotic and even some non-robotic tasks. We will discuss some of these applications in this section. Some applications use only the skill and sequencing tiers. Some applications use all three tiers. For each application we will describe the task, the robot, the skills, the RAPs, and, if applicable the plans. We will also give results and lessons learned from each application of the architecture.

3.1 A mobile robot that recognizes people

At the NASA Johnson Space Center's Robotic Architecture Laboratory, the 3T architecture has been used to control a robot that finds and recognizes people. The robot's task is to locate and approach a person wearing a specifically colored shirt, crop their face, feed the pixels of the cropped image to a neural network and then identify the person. The robot is a Cybermotion K2A base with a

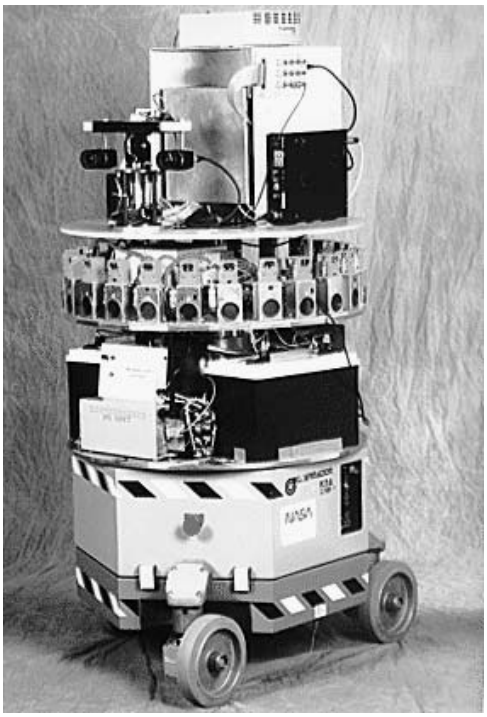

Figure 2: The Johnson Space Center robot programmed to find and recognize people. A color vision system is mounted on top of the robot; sonar sensors are just below it.

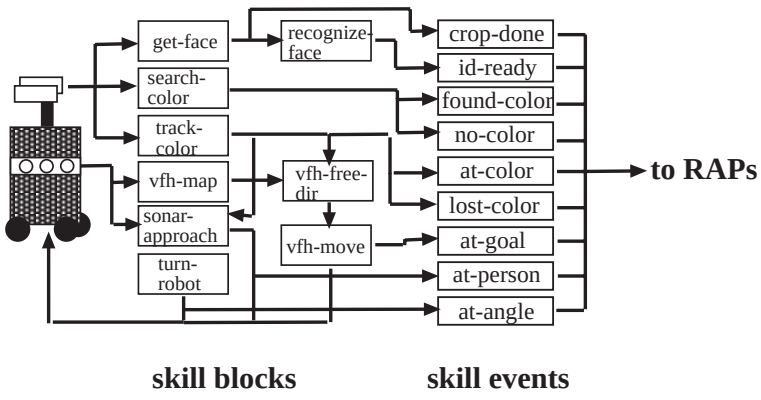

Figure 3: The skill network for finding and recognizing people.

ring of 24 sonar sensors and a color vision system mounted on top (see Figure 2. We will describe how this task was implemented in the bottom two tiers of our architecture, starting with the robot's skills. Details about the color vision system and the neural network can be found in [Wong *et al.*, 1995].

Figure 3 shows the skill network for this robot. These skills were implemented in C and executed entirely on-board the robot. The skills include visual skills for searching and tracking colors, a skill for cropping a face and a skill for recognizing

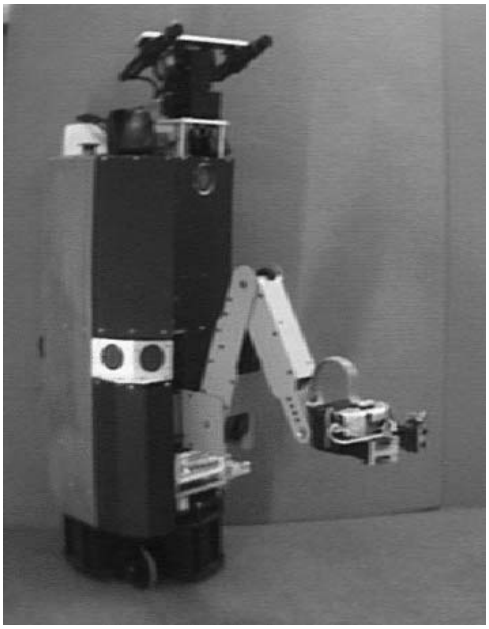

Figure 4: Chip with color vision system and manipulator

a face. There are also skills for moving the robot using obstacle avoidance (the VFH obstacle avoidance method [Borenstein and Koren, 1991]), approaching a person using sonar and turning the robot. Skill events report changes in the robot's state and in the state of the world to the RAP system. These include signals as to the state of the color searching and tracking, the state of face recognition and the state of the robot's motions.

The recognition task consists of several subtasks that are accomplished by activating sets of skills and waiting for an event (or events) to trigger. This application consists of about 20 RAPs that are responsible for enabling and disabling skill sets in order to accomplish the task and recover from errors. This application makes extensive use of the RAP context and method structures. Search RAPs cause the robot to move about our laboratory searching for a color. A history of the search is maintained in RAP memory so the robot avoids looking in places that it has already searched.

This application did not use the planning tier of the architecture. However, it does show how the bottom two tiers can perform a sophisticated task over a long period of time (the robot searched for up to four colors over thirty or forty minutes). The multitude of things that could go wrong (losing a color, not recognizing a person, inability to attain the goal location, etc.) demonstrated the recovery mechanisms built into RAPs. We would often "trick" the robot by flashing the color for which it was searching, then hiding it again and watching the robot stop, investigate the color and then continue the search when it could not locate the color again. In a similar application, the same robot equipped with a black and white stereo vision system used the bottom two tiers of the architecture to find and pursue other agents [Huber and Kortenkamp, 1995].

3.2 A trash-collecting mobile robot

As part of the Animate Agent Project at the University of Chicago's Department of Computer Science, the robot Chip has been programmed to clean up trash from the floor. Chip is an RWI robot with sonar sensors, a color vision system and a manipulator (see Figure 4). The color vision system is used to find and identify trash and trash bins. The manipulator is used to pick up and deposit the trash. This task was implemented using only the bottom two layers of the architecture. More details of this implementation can be found in [Firby *et al.*, 1995; Firby, 1995].

Chip uses a network of skills that include routines for moving in a given direction while avoiding obstacles, turning to face a particular direction, finding an object visually, tracking an object, and reaching toward an object. These skills, which are very general in nature, can be combined in various contexts to perform simple actions like: find a piece of trash, track a piece of trash and move towards it, align with a piece of trash, and pick a piece of trash up. By tracking the trash while approaching and aligning with it, the system can compensate for errors in the robot's motion and errors in initial estimates of the trash item's location. Skill events also report changes in the robot state and in the state of the world to the RAP system. These skills are written in LISP and C.

Sets of skills are enabled by the middle tier of the architecture. In combination these skill sets perform tasks such as moving to a location, tracking an object, aligning with an object, picking up an object and dropping an object. RAPs for these low-level tasks are then combined using higher-level RAPs to describe cleaning methods at various levels of abstraction as well as methods for moving the robot from one place to another, searching for an object, picking an object up, and putting an object in the trash. The RAPs also consult with a spatial planning module that keeps track of the floor area cleaned so far, and the locations of pieces of trash that have been seen but not yet dealt with.

The planner was not used in this application and all of the skills and RAPs are implemented on the Chip robot. As with the previous example, this application uses the bottom two tiers of the architecture to perform a complex task over a long period of time. Over many trials, Chip has picked up hundreds of pieces of trash and run for several hours without difficulty.

3.3 A mobile robot that navigates office buildings

At the MITRE Autonomous System's Laboratory, the 3T architecture has been used to program a robot to navigate the hallways and elevators of an office building. The robot is a Denning with a ring of 24 sonar sensors and a reflective barcode reader. The robot uses sonar data for obstacle avoidance and a laser scanner with bar-coded tags for landmark recognition. The laser landmarks consist of two coded tags. The laser scanning system returns only the angle to the tags in the robot's coordinate system. Using these angles in combination with a rough estimate of the distance between the tags allows the robot to roughly determine the distance and direction to the landmark when both tags are visible. Figure 5 depicts this domain. More details about this implementation can be found in [Firby and Slack, 1995].

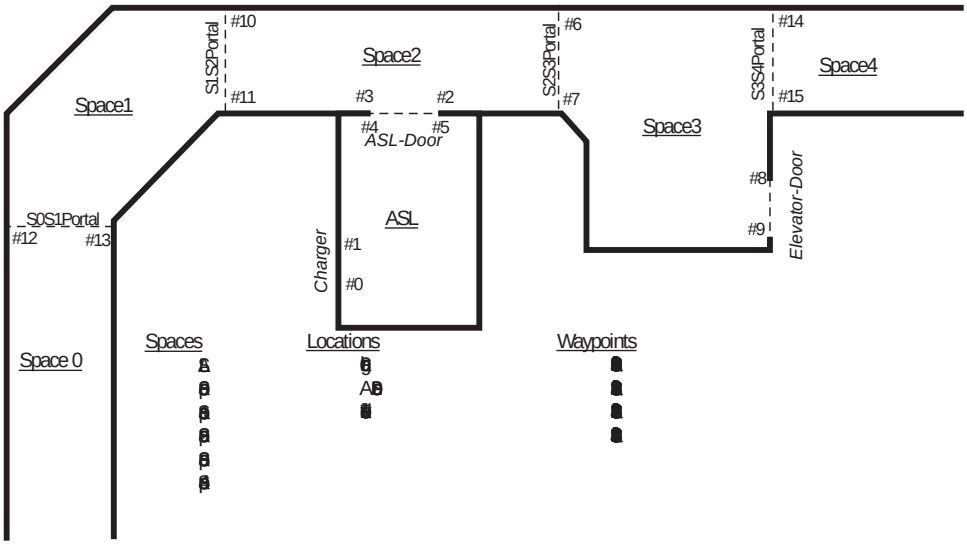

Figure 5: The MITRE navigation task domain

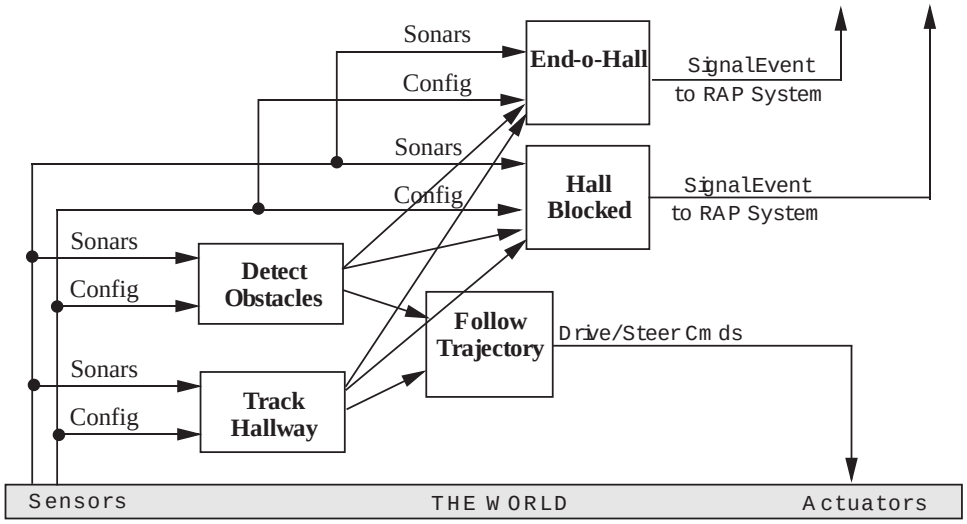

Figure 6: Partial skill network for the MITRE navigation task

Figure 6 shows a couple of skills for the navigation task. Additional skills included watching for landmarks, moving to a landmark, and moving through doorways. The primary navigation skills are Navigation Template-based (or NaT-based) [Slack, 1993] processes that use sonar information to avoid obstacles while moving to a given coordinate or moving in a particular direction. When constructed properly, such skill networks allow the robot to follow halls and enter doors independent of the specific hallway or door that the robot encounters. Skill events (e.g., HALL-BLOCKED in the Figure 6) report changes in the robot's state and in the state of the world to the RAP system.

Using these skills, RAPs were built to do tasks such as moving to a landmark,

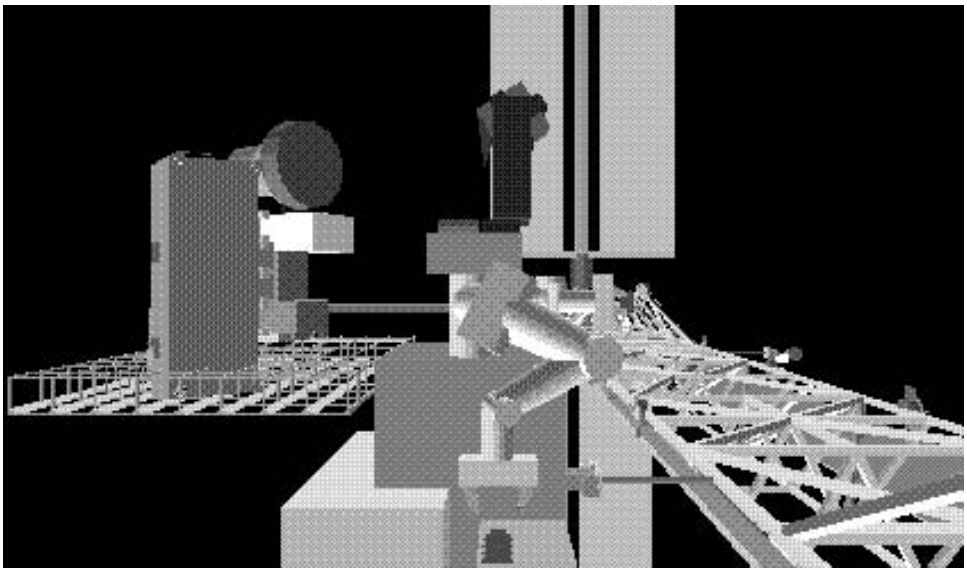

Figure 7: A simulated space station maintenance robot

moving to a neighboring space and moving through a set of connecting spaces. In addition, this application used the planning tier of the architecture to construct a plan to find its present location, plan a path to the elevator, navigate out of the room, through the door, down the hall, and up to the elevator. The planner enables the system to replan the robot's path if a hallway or doorway is blocked and to evaluate 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 can immediately abandon that goal and return to report failure.

3.4 Space station robots

Recently, the 3T architecture has been used as a framework for ground control of a two-armed manipulator robot maintaining a space station on orbit. The idea is for a manned, intelligent ground control station to supervise the routine maintenance activities of the robot, and allow the on-orbit personnel to concentrate on scientific missions.

In this application, the robot is a 3D kinematic simulation of a three-armed EVA Helper/Retriever (EVAHR) robot (see Figure 7) carrying out maintenance tasks around a space station, much as described in the introduction of this paper. A list of sites to be inspected or repaired is presented to the planning tier by the ground control supervisor and a maintenance plan consisting of repairs and inspections is generated for the day. The planner then installs each plan step on the RAPs agenda, along with the recommendation of the agent to use in the step (arms, cameras, etc.).

In the 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

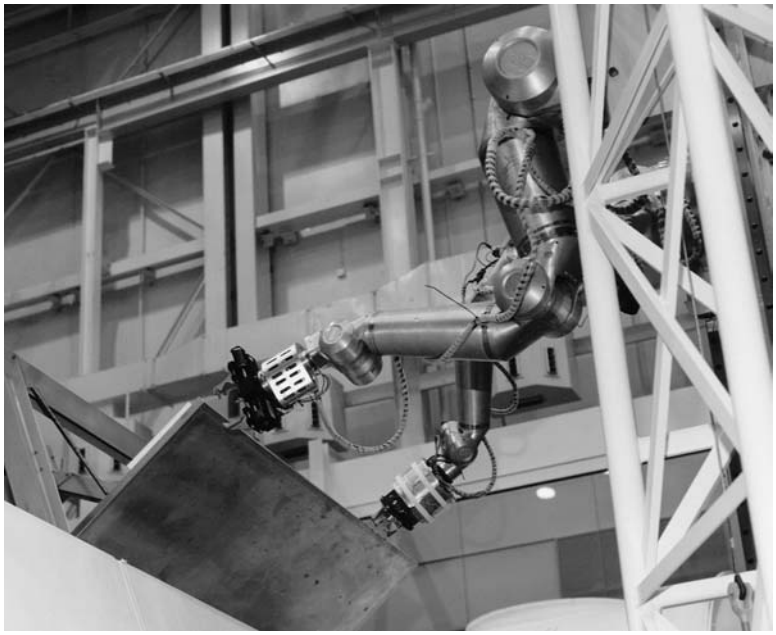

Figure 8: A prototype space station maintenance robot

future sites. An example concerns power-data attach points. EVAHR will move to the attach point which will afford the robot the most efficient mechanical access to the item being repaired. When that attach point malfunctions, the robot doesn't discover it until it tries to attach. The controlling RAP then consults a list of alternative attach points, selects one and moves to it. In some cases, the planner notices that this new point is also a convenient access point for the item in the next task. Thus, it will omit the next move operator in the plan and command only the next load, unload or inspect operator for the next item.

Should an arm fail, the RAP level is able to substitute an alternative agent (e.g., left arm instead of the recommended right arm) and this will cause the planner to adjust agent assignments for future tasks. 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.

After implementation on the simulator, the planner and RAPs system was ported to a hardware manifestation of such a service robot in a dual-armed facility known as ARMSS (Automatic Robotic Maintenance of Space Station). ARMSS consists primarily of two 7 degree-of-freedom Robotic Research manipulators, each mounted and moveable on vertical towers which in turn can move horizontally via a floor gantry (see Figure 8). These arms can reach a portion of a space station truss on which are mounted a variety of components that can be inspected or repaired. Camera views are available from each end effector and from the towers and the floor.

What was of interest in this project was construction of the reactive tier of the architecture out of an existing suite of software which emulated an actual

two-armed robot planned for space station Alpha. Existing ARMSS modules included inverse kinematic trajectory generation, velocity and position sensing and control, force/torque sensing and control, and ratchet and gripper control. These software modules could be accessed via the TelRIP Ethernet [Graves *et al.*, 1993] communications protocols. Thus, the skill system was simply functions that called and received signals from those modules. RAP primitives were encoded to enable and disable those skills and the RAPs from the EVAHR simulation were used to command the ARMSS system to changeout space station items. In this manner, two people were able to command the arms from RAPs within two weeks time. In a month the same two people had a previously teleoperated facility autonomously executing item changeouts as commanded by the AP tier.

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 and the RAPs were essentially unchanged.

4 Allocating knowledge across the architecture

The core 3T 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 on 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.[2] 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 other skills depending on its answer will be slowed down unduly. 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 very fast and often carries a lot of data, like images, sonar values, and real-time tracked-target position updates. On the other hand, the interface between the skill system and the RAP system consists primarily of commands to enable and disable skills and signals that certain skill-based events have occurred. This restricted bandwidth interface allows a very modular connection and easy distribution of the various tiers of the architecture across different machines (*i.e.*, the connection between skills and RAPs can be implemented via TCP/IP). Thus, skills are generally written to abstract perceptual information so that only small amounts of data

[2] In the actual described applications, the skills executed anywhere from 1/2hz to 10hz, depending on the implementation. These speeds were always more than sufficient for the described tasks.

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 nature of reactive skills requires that the possible connections between skills be specified in advance. Furthermore, skills must run very quickly and typically must be compiled in run-time networks. In contrast, the RAP system and planner are both based on interpreters and their behavior can be easily changed by adding or modifying RAP descriptions and planning operators. When a certain routine will require on-line modification by a human operator, then it should be put at the sequencer or planner level, not at the skill level.

5 Comparison With Other Work

Autonomous agent architectures fall into two broad categories: those designed from the outset to control physically embedded agents, and those initially designed to explore issues in general intelligence and later adapted for controlling physical agents. Examples of the former sort include subsumption [Brooks, 1986], TCA [Simmons, 1990], and AURA [Arkin, 1989]. Examples of the latter include SOAR [Laird *et al.*, 1987], and the architecture used in the Guardian program [Hayes-Roth, 1995]. We consider 3T to be an example of the former sort, that is, an architecture designed from the outset to control physical agents, and mobile robots in particular. Despite a large overlap between these two areas of research, we believe that there are certain issues that are unique to physical agents. For example, mobile robots can be expected to have stricter limitations on the amount of computing power at their disposal. Mobile agents may also face serious consequences if certain deadlines imposed by the environment are not met (though applications such as Guardian, which monitors patients in intensive care, face issues of similar gravity).

5.1 Robotic architectures

We begin with subsumption, arguably the best-known departure from the traditional sense-plan-act paradigm. The subsumption architecture is based on the idea of decomposing the problem of robot control by task rather than by function. Most architectures (including, to an extent, 3T) decompose the problem into functional modules such as planning, sensor processing, execution monitoring and contingency response. Brooks argues that such designs are inherently inefficient

because they force each functional module to be powerful enough to support any task the robot may be called upon to perform.

Rather than develop general functional modules, the subsumption architecture advocates the development of more narrowly focused mechanisms called behaviors. Each behavior is designed to control only a single task, allowing the computation within the behavior to be optimized for that task. Each behavior is coupled directly to the robot's sensors and actuators. Conflicts among behaviors are resolved by an arbitration mechanism.

Subsumption, and numerous variations on the theme (e.g. [Payton, 1990]), are all homogeneous architectures. The structure of behaviors and the way in which behaviors interact is the same throughout the architecture. There is no architectural support for abstraction, planning, or resource management. In fact, the design philosophy underlying subsumption specifically calls for such features to be avoided.

We believe that the subsumption position on plans and abstraction is too extreme. Although we agree with the motivating premise that there are serious shortcomings in the traditional sense-plan-act approach when applied to embodied agents, we believe that these problems can best be solved by changing the way in which plans are represented and used, not by discarding them entirely.

A three-tiered derivative of subsumption, SSS, was developed by Connell [Connell, 1992]. Subsumption makes up the middle tier of SSS, not, as one might suppose, the bottom tier, which is a collection of traditional servo-control loops. The ability of SSS to respond to contingencies is therefore limited by subsumption's finite-state-machine model. SSS adds a "contingency table" representation, making programming somewhat less cumbersome than constructing FSA's directly, but it is still quite restrictive. SSS has only been demonstrated on tasks involving pure navigation (although it performs this task with impressive speed).

Simmons' Task Control Architecture (TCA) [Simmons, 1990] has been successfully used on a number of real-world robots, but it is very different from 3T. There are essentially no tiers in TCA. A task net is constructed for the robot which is similar to a task-net in RAPs. Each node in the task tree can be decomposed further or is a primitive which interfaces with the robot, or other nodes, through a sophisticated message-passing algorithm. These messages are processed through a central router, and thus TCA is more like a robot operating system. There are no explicit representations for expressing relationships among tasks. TCA task trees are manipulated directly by C function calls. It is therefore incumbent on the programmer to mentally compile the desired control constructs into the appropriate calls.

Furthermore, because TCA lacks a representation for task trees it is cumbersome to employ a general planner, since the output of the planner has to be translated into C code and compiled, or an interpreter that translates the output of the planner into TCA calls has to be written. This may not be a difficult task, depending on the planner, but it is not a generalized procedure as is the use of the planner in 3T.

Finally, the view of cognizant failure is quite different in TCA. Since the only place to put knowledge of failure is in the task nets, in TCA one builds exception

subnets as add-ons to the "normal" task trees. 3T on the other hand deals with failure at three levels: environmental variation in the skills, variation in routine activity in the RAPs and variation in time and resources in the planner.

Of the robot architectures in use, 3T has its strongest similarity to ATLANTIS [Gat, 1992]. ATLANTIS and 3T grew out of the same work – hence their similarity. The chief differences between the two is that ATLANTIS leaves much more control at the sequencing tier. In ATLANTIS, the deliberative tier must be specifically called by the sequencing tier.

Noreils [Noreils and Chatila, 1995] describes an architecture that integrates planning and reactivity. Noreils' architecture has three levels (called planning, control, and functional) which correspond roughly to the three main components of 3T. The control level is based on a formalism that is similar to RAPs. The principal difference between Noreils' formalism and RAPs is that Noreils' formalism distinguishes between failures and non-failures, whereas the RAPs ontology simply considers multiple outcomes without requiring them to be further categorized. We believe this is an important distinction because as the tasks a robot performs become more complex the division between failure and non-failure can become very fuzzy, and requiring the user to make this distinction can become burdensome. Noreils's architecture has been implemented on two robots, with some of the computation being done off-board.

There are numerous architectures which are specific to mobile robot navigation. AURA [Arkin, 1989] is superficially similar to 3T. It based on a fundamental building block of a motor schema, a vector field associated with a goal or an obstacle. Motor schemas are combined by vector addition to produce a resultant which drives the robot. A second "tier" enables combinations of these vector fields to run concurrently to produce the desired effect. But this second level lacks many of the context based reordering capabilities of RAPs. This is because AURA is concerned with generating a world model at this second level that is connected directly to the low level motor schemas. In 3Tthe sensing from the control tier uses explicit task context to determine its meaning to the rest of the architecture.

The control community also has some noteworthy multi-tiered architectures. NASREM [Albus *et al.*, 1986] was an early reference model for telerobotic control. It was in essence a many tiered model which predated 3T in its provision for increasing abstraction and increasing cycle time as it moved from the servo level to the reasoning levels. With the exception of maintaining a global world model, NASREM, in its original inception, provided for all the data and control paths that are present in 3T. But NASREM was a reference model, not an implementation. The subsequent implementations of NASREM followed primarily the traditional sense-plan-act approach and were mainly applied to telerobotic applications, as opposed to autonomous robots. A notable exception was the early work of Blidberg [Blidberg and Chappell, 1986].

Saridis' intelligent control architecture [Saridis, 1995], while having three layers, is fundamentally different from 3T in its philosophy and implementation. The architecture begins with the servo systems available on a given robot and augments them to integrate the execution algorithms of the next level, using VXWORKS and the VME-bus. The next level consists of a set of coordinating

routines for each lower subsystem, e.g., vision, arm motion, navigation. These are implemented in Petri Net Transducers (PNTs), a kind of scheduling mechanism, and activated by a dispatcher connected to the organizational level. The organizational level is a planner implemented as a neural network of the Boltzmann variety. Essentially the neural network finds a sequence of actions which will match the required command received as text input, and then the dispatcher executes each of these steps via the network of PNT coordinators.

The emphasis in this architecture is increasing precision with decreasing intelligence (IPDI). This sounds vaguely like the abstraction hierarchy of 3T. But the intelligence of the system stems from 1) probability models of task decomposition and execution, and 2) functions that minimize a measure of entropy at each layer. The result seems to be sequences of actions – from the neural network and then from the coordinators – that if executed will have the highest probability of success.

The IPDI architecture has only been implemented on a PUMA manipulator robot and then apparently only partially so. Thus it is difficult to judge the relative merits of a neural network planner versus a symbolic planner or the RAPs system versus a dispatcher of Petri Net coordinators without a detailed analysis which is beyond the scope of this paper. It appears however, that IPDI contains little if any provision for a dynamically reconfigurable skill network, cognizant failure, or recovery from such failures at every level. As well, it also appears that IPDI is currently only applicable to robots using VXWORKS and the VME-bus.

5.2 Non-robotic agent architectures

Many architectures have been proposed for controlling intelligent agents. We focus here on those that have been applied to controlling physical robots.

The Guardian architecture of Hayes-Roth is a blackboard architecture designed for controlling embedded (though not necessarily embodied) agents. The architecture is divided into a cognitive component and a perception/action component. The perception/action component is controlled by the cognitive component. Thus, the Guardian architecture is similar to 3T, but with sequencing and deliberation performed by the same mechanism. The deliberative component can modulate the performance of the perception/ action component, as well as its own performance, according to the current situation in the world. Guardian representation, but does not commit to any particular representation for describing the interrelationships among tasks.

The major capability of the architecture is the ability to migrate decision-making from the slow deliberative component into the faster perception-action component by reasoning about the current situation. For example, Guardian can reason about the types and sampling rates of perceptual tasks that need to be performed to support a given goal. 3T, by contrast, takes the position that rather than reason about which control mode is appropriate according to the task environment, that *both* modes must be operating constantly no matter what the environment. The appropriate adaption arises automatically when the sequencer activates and deactivates skill sets according to the current situation,

while the planner reasons about the overall goal by projecting alternative futures. 3T does not explicitly distinguish between perceptual tasks and non-perceptual tasks; perceptual tasks are treated like any other task. Perceptual tasks produce results (information) that can be preconditions of other tasks, but this is all handled with one unifying mechanism. We find that even in very complex cases, the appropriate behavior arises naturally from encoding tasks in the RAP representation.

Soar [Laird *et al.*, 1987] is a production system with the ability to switch between deliberative and more reactive modes of reasoning via a learning mechanism that caches deliberative results. Soar has been augmented with a perceptual-motor interface [Weismeyer, 1989] which is constructed from the same basic computational mechanism as the rest of the system. Soar thus collapses all of the capabilities of 3Tinto a single mechanism. Like Guardian, Soar embraces the concept of representation, but does not commit to a particular task representation.

3T shares many aspects of Cypress [Wilkins *et al.*, 1995]. Our AP planner has similar expressive power at an abstract level as 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 specify a canonical interface to the control tier as does 3T.

CIRCA [Musliner *et al.*, 1993] has been used only with simulations, but 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, though for the record, none of the tasks described in the previous section required such a capability.

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. We have found that the division of labor among the tiers 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 believe that 3T can ease the development of software control code, which is notoriously complex, on a wide variety of robot systems. 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, 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.

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 tier 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 robots search 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 managing closed ecological life 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 tiers 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 also begun to investigate the integration of other AI disciplines. Natural language is already being researched at the RAPs level [Martin and Firby, 1991]. Machine learning techniques can be investigated from case-based reasoning in the planning tier to reinforcement learning in the skill tier [Bonasso and Kortenkamp, 1994]. The architecture could also benefit from combining it with concurrent perception architectures such as those used to support mapping [Kuipers and Byun, 1991; Kortenkamp and Weymouth, 1994].

References

[Albus *et al.*, 1986] J.S. Albus, R. Lumia, and H.G. McCain. Nasa/nbs standard reference model for telerobot control system architecture (nasrem). Technical Report Tech Note #1235, NASA SS-GFSC-0027, National Bureau of Standards, 1986.

[Arkin, 1989] Ron Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4), 1989.

[Blidberg and Chappell, 1986] D.R. Blidberg and S.G. Chappell. Guidance and control architecture for the eave vehicle. *IEEE Journal of Ocean Engineering*, OE-11(4):449–461, 1986.

[Bonasso and Kortenkamp, 1994] 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.

[Bonasso *et al.*, 1992] 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.

[Borenstein and Koren, 1991] Johann Borenstein and Yoram Koren. The Vector Field Histogram for fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3), 1991.

[Brooks, 1986] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

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

[Elsaesser and MacMillan, 1991] Chris Elsaesser and Richard MacMillan. Representation and algorithms for multiagent adversarial planning. Technical Report MTR-91W000207, The MITRE Corporation, 1991.

[Elsaesser and Slack, 1994] 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.

[Firby and Slack, 1995] R. James Firby and Marc G. Slack. Task execution: Interfacing to reactive skill networks. In *Working Notes: 1995 AAAI Spring Symposium on Lessons Learned from Implemented Architecture for Physical Agents*, 1995.

[Firby *et al.*, 1995] R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain. An architecture for vision and action. In *International Joint Conference on Artificial Intelligence (to appear)*, Montreal, Canada, August 1995. IJCAI.

[Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1987.

[Firby, 1989] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds.* PhD thesis, Yale University, 1989.

[Firby, 1995] R. James Firby. Lessons learned from the animate agent project. In *Working Notes: 1995 AAAI Spring Symposium on Lessons Learned from Implemented Architecture for Physical Agents*, 1995.

[Gat *et al.*, 1989] 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.

[Gat, 1992] 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.

[Graves *et al.*, 1993] Sean Graves, Larry Ciscon, and J.D. Wise. A telerobotic interface protocol. In *Proceedings of IEEE International Confernence on Robotics and Automation*, 1993.

[Hayes-Roth, 1995] Barbara Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72, 1995.

[Huber and Kortenkamp, 1995] 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.

[Kortenkamp and Weymouth, 1994] 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.

[Kortenkamp *et al.*, 1993] David Kortenkamp, Marcus Huber, Charles Cohen, Ulrich Raschke, Clint Bidlack, Clare Bates Congdon, Frank Koss, and Terry Weymouth. Integrated mobile robot design: Winning the AAAI-92 robot competition. *IEEE Expert*, 8(4), August 1993.

[Kuipers and Byun, 1991] 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.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1), 1987.

[Martin and Firby, 1991] 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.

[Miller *et al.*, 1994] 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.

[Miller, 1986] 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.

[Musliner *et al.*, 1993] David J. Musliner, Ed Durfee, and Kang Shin. CIRCA: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993.

[Noreils and Chatila, 1995] Fabric Noreils and Raja Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, 2, 1995.

[Payton, 1990] David Payton. Exploiting plans as resources for action. In *Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, CA, November 1990. DARPA.

[Saridis, 1995] G.N. Saridis. Architectures for intelligent controls. In Gupta and Sinhm, editors, *Intelligent Control Systems: Theory and Applications*. IEEE Press, 1995.

[Shoham, 1988] Y. Shoham. *Reasoning about Change*. MIT Press, Cambridge, MA, 1988.

[Simmons, 1990] Reid Simmons. An architecture for coordinating planning, sensing and action. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990.

[Slack, 1992a] Marc G. Slack. Computation limited sonar-based local navigation. In *Proceedings of the AAAI 92 Spring Symposium on Selective Perception*, 1992.

[Slack, 1992b] Marc G. Slack. Sequencing formally defined reactions for robotic activity: Integrating raps and gapps. In *Proceedings of SPIE's Conference on Sensor Fusion*, 1992.

[Slack, 1993] Marc G. Slack. Navigation templates: Mediating qualitative guidance and quantitative control in mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 23(2), 1993.

[Weismeyer, 1989] Mark Weismeyer. New and improved Soar I/O. Technical report, University of Michigan, 1989.

[Wilkins *et al.*, 1995] 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.

[Wong *et al.*, 1995] 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.

[Yu *et al.*, 1994] 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.