Issues in Providing Adjustable Autonomy in the 3T Architecture

Pete Bonasso

Texas Robotics & Automation Center Laboratories (TRACLabs)
1012 Hercules
Houston, TX 77059
bonasso@aio.jsc.nasa.gov

Introduction[1]

Three-tiered robot control architectures, such as 3T, Atlantis and the Remote Agent Architecture, have been applied, in some incarnation to dozens of field, service, and space robots and other computer controlled machines for almost a decade (see the Architectures section in [Kortenkamp et al 1997]. In every case, including the Deep Space 1 application, some attention has had to be paid to interacting with the human creators, programmers and users of these robots. In the past five years, since 1994, the developers of 3T at TRACLabs have experienced a variety of needs for adjusting the level of autonomy in the control of these machines in various applications at the Johnson Space Center. Beginning in late 1995, we began a research program to begin to put our ad hoc experiences into a set of principles embodied in adjustable autonomy (AA) software tools for the architecture. We have seen the need for humans to be involved not only at the obvious deliberative layer, but all the way down to teleoperating what were supposed to be machines running autonomously. Moreover, we have found that one guiding principle seems to put all of our AA endeavors in a unified light: design the computer controlled machine and it's control architecture for full autonomy; then relax the autonomy restriction at each level, beginning with the highest.

This paper will detail a number of issues and solutions that we have been dealing with in applying that basic AA principle to 3T and it's attendant applications. The applications have driven us to try to develop additional AA capability, while that development has made us look deeper for more general principles and approaches to the problem of humans and computer controlled machines working in harmony to accomplish a mission.

Adjustable Autonomy at Each Tier

We discuss AA issues and some of our solutions at each tier of the architecture (Figure 1). We will use as a focusing application, a pick and place robot used for inspecting and replacing experiments on a task board in a bay outside the space station. Figure 2 shows a typical display for operating at the first two tiers. A plan has been generated and is displayed as a hierarchical task net (HTN) with agent assignments (humans or the robot called, Chroma). Below the HTN there is also a Gantt chart in the background showing the schedule of actions. In the center of the figure is a small graphic depiction of the top view of the task board.

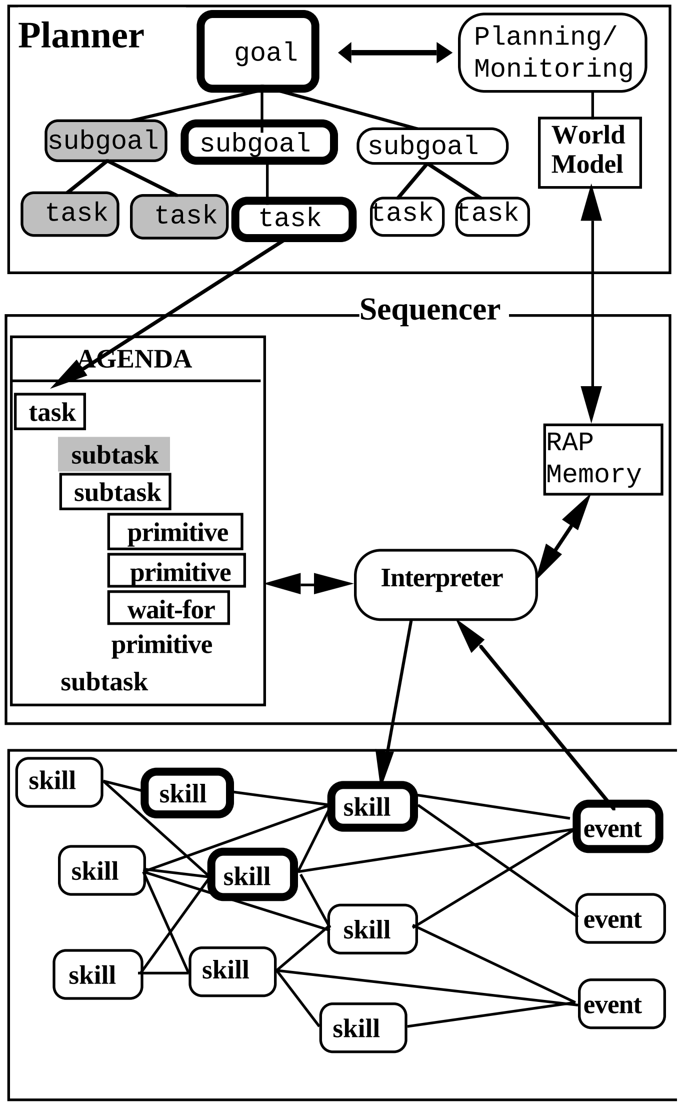


Figure 1. The 3T Architecture

[1]Copyright © 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

At the right is a user interface dialog box which indicates the types of AA interactions to be discussed.

In the pick and place scenario, humans and robots cooperate to carrying out the required functions. For example, the robot (the long, thin rectangle in the graphic) can move experiments and around and tighten bolts but cannot manipulate the protective blanket (the large square in the graphic). Likewise, the robot can bring an experiment in view of the camera (the small box over the arm in the graphic), but the human makes the ultimate decision whether or not to replace the experiment.

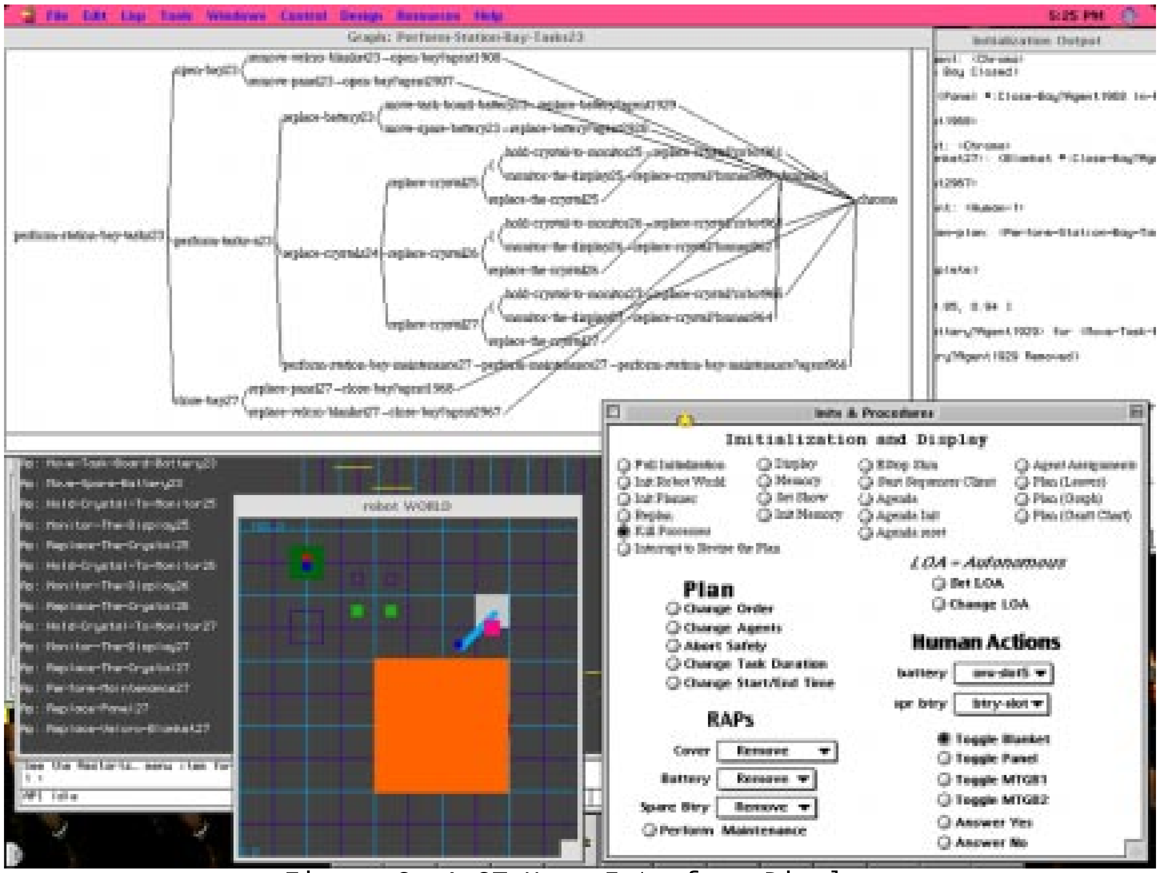
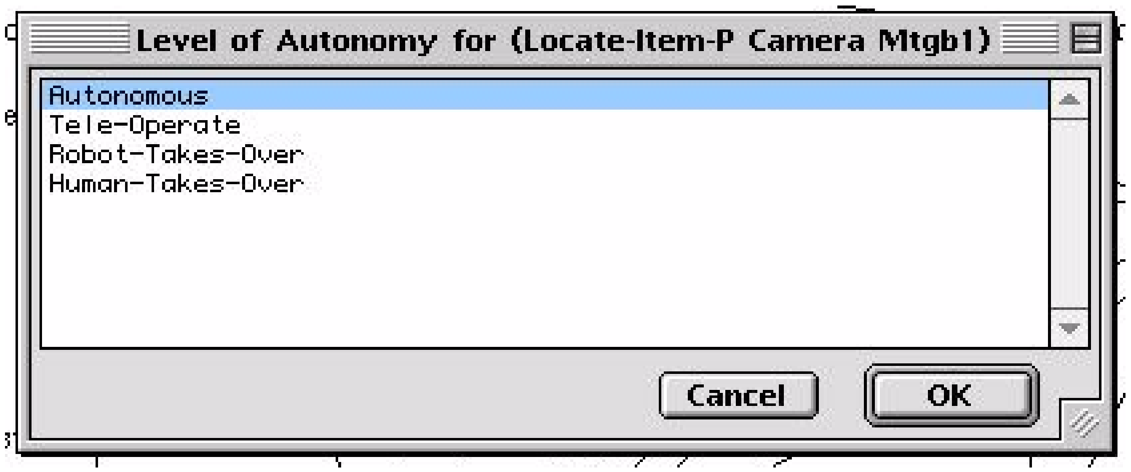


Figure 2. A 3T User Interface Display

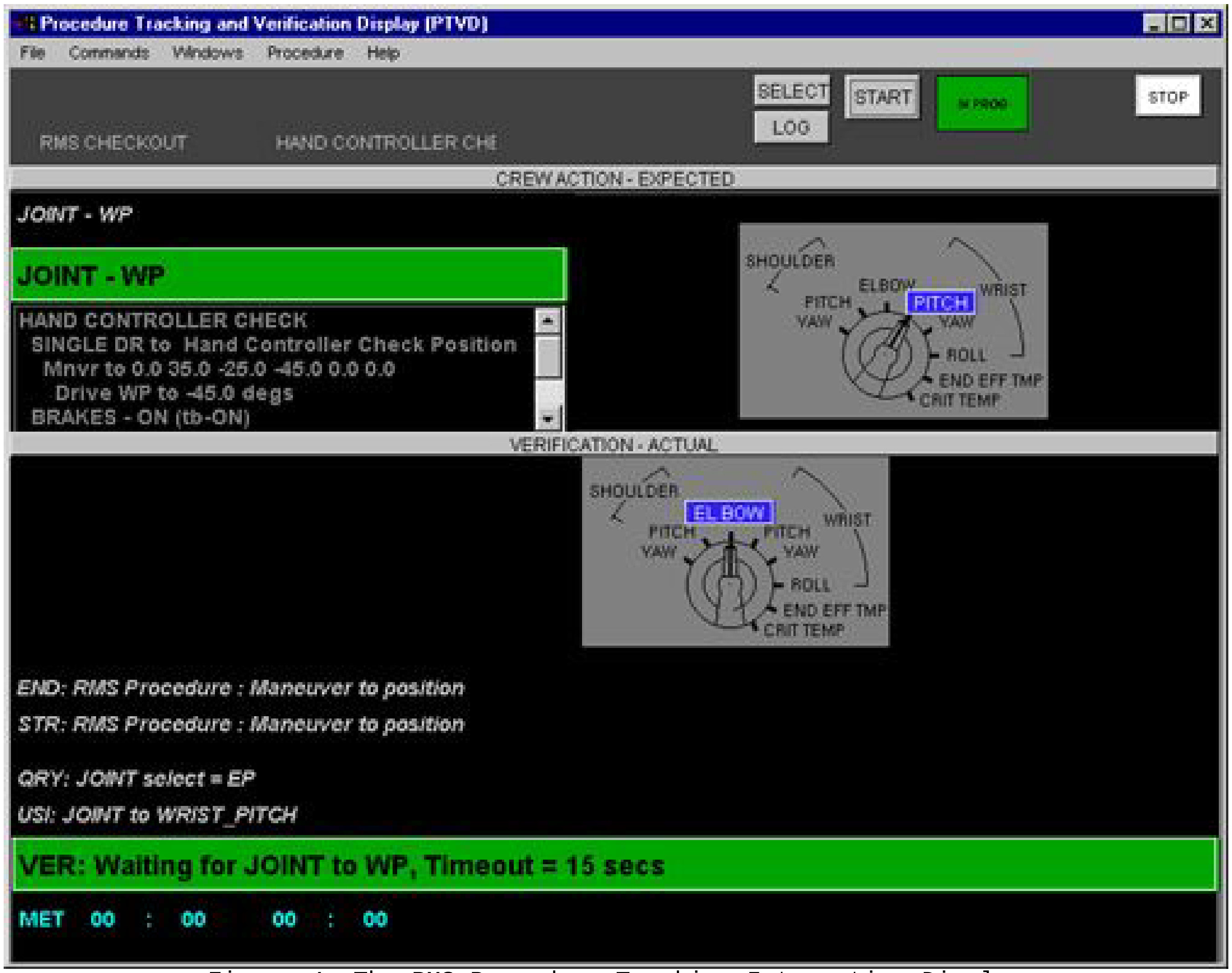


Level of Autonomy for (Locate-Item-P Camera Mtgb1)

[✓ Autonomous]
Tele-Operate
Robot-Takes-Over
Human-Takes-Over

Cancel OK

Figure 3. A Level of Autonomy Refinement Query



JOINT - WP

JOINT - WP

HAND CONTROLLER CHECK
SINGLE DR to Hand Controller Check Position
Mnvr to 0.0 35.0 -25.0 -45.0 0.0 0.0
Drive WP to -45.0 degs
BRAKES - ON (tb-ON)

END: RMS Procedure : Maneuver to position

STR: RMS Procedure : Maneuver to position

QRY: JOINT select = EP

USI: JOINT to WRIST_PITCH

VER: Waiting for JOINT to WP, Timeout = 15 secs

Figure 4. The RMS Procedure Tracking Interactive Display

AA at the Deliberative Layer

At the deliberative layer, we have been looking at not only planning for the actions of humans in joint human/robot tasks, but also the mixed initiative requirements necessary to bring human preferences to bear on the outcome. In our instantiation of 3T, we use the Adversarial Planner (AP) [Elsaesser & Slack 94] as our deliberative tier. The deliberative tier of the architecture is responsible for coming up with a plan of action to accomplish a high-level goal, including agent assignments for each primitive action. In our scenario, where the human and robot cooperatively work together to perform specified tasks in the station bay, the goal is to change out experiments that are ready to change out. Secondary goals include inspecting the battery and tightening any loose maintenance bolts. Once a plan is published, AP monitors its execution to ensure that at each step the plan is still valid, and can replan if necessary. In our AA research we take this a step further: during execution monitoring a user can interrupt the planner and make changes/recommendations to the planner, and the planner will replan, respecting the user's wishes as much as possible. This mixed-initiative planning allows a user to interact with the planner to influence the resulting plan, yet allows the system to insure hard constraints are not violated.

For the station bay scenario we modified AP to respond to user input when creating a plan. Basically, the user can now specify which agents should perform the various tasks, the order that the tasks should be performed, the expected duration of a task, or can stop the execution of the plan at any time and close down the station bay. There are two times when the user may want to affect the plan. Before plan generation the user can specify preferences for agents and ordering (for partial orders), and can input updated expected task times. During plan execution the user can interrupt the system and modify agents, ordering and execution cessation. These modifications are viewed as preferences - any hard coded safety rules have priority. By adding these capabilities to the planner, AP now responds to user commands as an integral part of its execution monitoring.

In another of the NASA applications where the deliberative tier was used, AP had to plan for the activities in a closed, regenerative life support system 90 day test [Schreckenghost et al 98]. Here the mixed initiative revolved around time. The users wanted 16, 20 or 24 day plans, depending on various crop growth rates (crops provided oxygen to the crew of four). But they also wanted a new plan every four days. This was because though solid wastes were nominally incinerated every four days, often there was not enough waste for a full burn and the user wanted the option of deciding when to conduct the incinerations (the incinerator was also a consumer of plant generated oxygen).

AA at the Conditional Sequencing Layer

It is at the conditional sequencing layer that we have developed the most software tools for mixing human and robot control -- we can seamlessly invoke the continuum from shared control (e.g., sharing degrees of freedom) to traded control, where the robot and the human actually take over control of the task from the other at various stages of the execution.

The 3T sequencing layer is written in the Reactive Action Package System (RAPS) [Firby 97]. Each primitive operator of the planner corresponds to one or more RAPs in the sequencer's RAP library. Each RAP represents a parameterized linear plan which can be decomposed into other RAPs or which can invoke the reactive behaviors of the skills level.

The general adjustable autonomy approach at this level is three fold. First, each primitive RAP has associated with it a level of autonomy which is set by the user at the outset and normally defaults to semi-autonomous. That is, each time the primitive is invoked, users are queried as to whether they want the operation done by the robot or by the human. There are also two other choices : the human can take over and do everything in a teleoperated mode or the remainder of the tasks are given to the robot to complete autonomously. The user can reset this primitive LOA mode at anytime.

The second area of RAPs development concerning adjustable autonomy has to do with agent specification. The index for a RAP includes an agent variable which is set by the planner as part of its agent assignment function. If the agent specified for any RAP above the primitives is a human, the LOA for primitive RAPs is set to tele-operation, otherwise, the LOA is set to autonomous. But sometimes, the user needs to be able to override that assignment, for example, to make fine adjustments to a robot manipulator position in the viewing camera. Thus, once a higher level RAP begins executing, the user can change the LOA via the user interface.

The third area to effect autonomy in RAPs involve RAPs that can only be done by a human or a robot because of safety or the physical constraints of the job. For example, our robot is not dexterous enough to remove or replace the blanket over the work bay. The primitive concerning this RAP as well as any higher level RAPs that invoke it, uses an ?agent designation from the planner but does not invoke the LOA query.

We have learned that since the robot needs monitoring capabilities to be able to function autonomously, that same monitoring can track the human doing any part of the same task. The example RAP below, shows how this is done. Using the level of autonomy (LOA) query, a method is selected corresponding to the current default -- autonomous, semi-autonomous, or tele-operate. The semi-autonomous LOA further queries the user to set the LOA for this step or for future steps (robot or human takes over) (see Figure 3), and ultimately returns autonomous or tele-operate for at least this one primitive invocation. Notice that the two methods are identical save for the enabled action. In the autonomous case, the action enabled is a robot primitive; in the tele-operate case, the action is simply to notify the user to move the robot's arm to the given place and orientation. The event processing is the same.

Of course the human can deviate at her discretion from the task procedures that were coded for the robot. So our toughest problem here is a variation of the frame problem: the

```
(define-rap (arm-move-p ?arm ?place ?orientation
               ?timeout)
   (succeed     (or   (and     (arm-at ?arm ?where ?res)
                               (= ?where ?place))
                               (arm-place ?arm ?place)))
   (timeout ?timeout)
   (method robot-move
      (context   (and     (LOA arm-move-p ?arm ?place ?loa)
                          (= ?loa autonomous)))
      (primitive
         (enable  (:arm_move   (:place . ?place)
                               (:orientation . ?orientation))
                  (post-to-log
                     (format nil
   "VER: Waiting for arm movement complete,
                            Timeout = ~a." ?timeout)))
         (wait-for
          (arm-move-done? ?arm ?place ?orientation ?result)
                 :succeed (arm-move ?result))
         (disable :above)))
   (method human-move
      (context   (and     (LOA arm-move-p ?arm ?place ?loa)
                          (= ?loa tele-operate)))
      (primitive
         (enable    (tell-user
            "~a, please move ~a to ~a." 'human '?arm '?place)
                  (post-to-log
                     (format nil
   "VER: Waiting for arm movement complete,
                            Timeout = ~a."  '?timeout)))
         (wait-for
            (arm-move-done? ?arm ?place ?orientation ?result)
                  :succeed (arm-move ?result))
         (disable :above))))
```

robot must ask itself "What has changed that I didn't notice since the human took over? Fortunately it is at this level that we can use the reactive execution capability of the sequencer to bring additional sensing to bear to verify the presence or absence of essential objects before continuing the task. In some applications we have opted for periodic camera scans of the work area. Since these can be time consuming and expensive, another approach we have used is to code a "locate" RAP which is invoked whenever an item needs to be used in an operation. If it has been awhile since the robot updated the item's location in memory, or if a lower level operation fails (possibly because the item is not where it was a few minutes ago), the locate RAP is invoked on the item to re-establish its current location (and in some cases, it's operational status).

Other methods we have considered involve using the task context to circumscribe the work area of concern, and having the robot engage in a dialog with the human to obtain any missing states that cannot be autonomously detected by the robot.

In our most involved AA application to date at NASA, the Remote Manipulator System (RMS) Procedure Tracking System [Bonasso et al 97], the use of the LOA became the fundamental basis of determining which procedures for a currently all manual system would have an autonomous option for the future.

An RMS procedure is documented as a set of steps in a flight data file (FDF). This is not a computer file but a small booklet tabbed with the appropriate test. A second crewman, known as the R2 crewman, stands behind the RMS operator, reading off the steps from the FDF. In teleoperated mode, the RMS Procedure Tracking System eliminated the need for R2 by providing both highlighted text and voice prompting cues for each step, a continuous update of the RAPs task agenda for context, and status messages for each step (see Figure 4). This was the only mode possible with today's RMS, which has no means for automatically commanding a procedure. But using a *simulation* of the RMS, complete with virtual sensors, for each task, an experienced astronaut could select that the task be done autonomously, and view the resulting effects. Over a period of time, the astronaut office would be able to determine not only which items in future FDFs could be carried out automatically, but also what kind of avionics upgrades would be needed to provide the necessary sensing data to the autonomous system

AA at the Reactive Layer

Recently, a number of surprises in AA applications have come at the skills or behavior level, when some systems that we expected would have no human involvement -- indeed had no seeming pragmatic reason for human involvement -- were required to have such involvement for purposes of subsystem testing or simply giving the humans confidence that the system would perform as specified. Yet in attempting to achieve this level of AA -- essentially seamless teleoperation in an autonomous system -- we found a number of powerful innovations which were only possible because the rest of the architecture was in place.

These changes were the result of using 3T to support tests for advanced air and water recycling life support equipment. These systems were to be run autonomously on the space station or in a planetary environment, and we constructed the 3T applications as such. We included the AA facets described elsewhere in this paper, but none at the skills level. But because of a tight schedule, these life support systems were being redesigned to meet space requirements even as they were executing in a "flight" test environment. This meant that the life support engineers would not know precisely what settings should be made such things as pump speeds when these redesigns occurred. So the engineers asked for the capability to be able to toggle switches and set pump speeds manually from time to time. The problem of course is that a human not versed in the 3T checks and balances could put the system in a state that would cause 3T to execute some emergency procedure or even shut the system down. The solution to this conundrum was to inform the rest of the architecture that a human was a possible wrench in the works.

As shown in Figure 1, the reactive level in 3T is a dynamically reconfigurable network of robot skills coordinated by a skill manager. For example, grasping, object tracking, and local navigation are robot skills. These are tightly bound to the specific hardware of the robot and must interact with the world in real-time. We develop robot skills in C code or use existing robot control algorithms. The skills are enabled or disabled by the sequencing tier of the architecture, usually in groups that represent a competence for a particular task. At least one of these skills is a device skill which ties directly to the A/D boards for the robot. This skill provides data to all the other skills and passes commands from those skills to the machine devices. To allow the human to drive any low level actuator directly we developed the notion of a "tele-op" skill. This skill has an output for each device tied to the device skill, and once activated, receives GUI commands for any of the machine devices, and passes them to the device skill.

So the normal course of events is that the user will decide to use the teleop capability and click on a 3T GUI button which sends a "teleop_start" message to the sequencer. The sequencer will enable the teleop skill, pop-up a special tele-op GUI, send a "teleop" parameter to the device skill and *post that fact in the RAPs memory*. Now whenever the user, via the teleop GUI, sends a device command, the device skill will use that command to override any 3T commands to the device as long as the teleop skill is active.

Since the sequencer knows that human teleop may be going on, it can add that fact to the context of any of its emergency procedures. This might take the form of simply putting out a warning message but taking no action. In most cases the user wants an interactive option to allow 3T to carry out the procedure. When the user allows 3T to take over, the teleop skill is first disabled.

Conclusions

This then is a consistent theme in our AA work. Our three layered control system (and we assume others like it) which has proven efficacious in autonomous control turns out to be the best kind of framework for embedding adjustable autonomy in all its forms. We have seen that the planner's execution monitor, designed for autonomous planning and replanning, can also monitor for "user states" which indicate the current plan is not acceptable just like any other sensed phenomenon. The sequencer employs a user variable to decide whether to enable a robot skill, but uses the same sensing skills that exist for full autonomy operations to track those user actions. Finally, the skills model of 3T allows 3T to seamlessly enable a "human robot" to take any and all low level actions, overriding built-in autonomous monitoring and reaction, until such time as the human wishes to resume normal adjustable autonomy operations.

References

Bonasso, R.P., Kortenkamp, D. and Whitney, T. 1997. Using a Robot Control Architecture to Automate Space Shuttle Operations, 9th Conference on Innovative Applications of AI (IAAI97) , July.

Elsaesser, Chris and Slack, Marc. 1994. Integrating Deliberative Planning in a Robot Architecture. Proceedings of the AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS '94).

Firby, James R. 1997. The RAPs Language Manual. Neodesic, Inc. Chicago.

Kortenkamp, D.K., Bonasso, R. P. & Murphy, R., eds. 1997. AI-based Mobile Robots, AAAI/MIT Press.

Schreckenghost, D; Bonasso, P.; Kortenkamp, D.; and Ryan, D. 1998. Three Tier Architecture for Controlling Space Life Support Systems . IEEE Symposium on Intelligence in Automation and Robotics. May.