S23-60
188105
11P

A Run-Time Control Architecture for the JPL Telerobot

J. Balaram, A. Lokshin, K. Kreutz, and J. Beahan
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

JJ574450

1. Abstract

This paper describes an architecture for implementing the process-level decision making for a hierarchically structured telerobot currently being implemented at the Jet Propulsion Laboratory (JPL). Constraints on the architecture design, architecture partitioning concepts, and a detailed description of the existing and proposed implementations are provided.

2. Introduction

The architecture of a telerobot is required to support autonomous and teleoperated activities in a manner designed to obtain the maximum synergy of function between robotics and teleoperation. The JPL telerobot implements such a system for applications involving various servicing and repair operations in space.

The architecture proposed for the JPL telerobot decomposes the functionality of system operations into three hierarchical levels. For the autonomous component of the architecture, these levels consist of the Task Planning Level at the top of the hierarchy, followed by the Process-Level at the intermediate level, and the Actuation and Sensor Level at the bottom of the hierarchy. Each of these levels is designed to operate robustly by using local feedback to detect and recover from local errors.

The Task Planning level is concerned with the overall context of the task, including global planning, overall execution monitoring and task replanning. The basis for the decision making is the Task Sequence Logic with relatively little incorporation of Task Domain Physics. As a consequence, activity timing at this level is driven by task scheduling requirements, and the decision making methods used are predominantly symbolic and not numeric. In the JPL telerobot the functions at this level are mechanized by an Artificial Intelligence Planner subsystem (AIP) with the logical planning of a Module swap-out sequence being a representative example of internal subsystem activity.

The Process Level is concerned with the planning, execution and monitoring of subtasks such as grasping objects, object assembly etc. Activities at this level require only a local subtask context within which various sensor and actuation subsystems are commanded and coordinated to accomplish a given subtask. Decision making at this level has to be cognizant of the physics and geometry of the task domain and the occurrence of various run-time physical events in the telerobot and its environment. Decision making is characterized by a hybrid of symbolic and numeric processing with activity timing determined by the event line of physical sensed events. Mechanization at this level is provided by a Run-Time Control subsystem (RTC) which performs the various trajectory determinations, reflex specifications etc. for subtasks such as grasp centering and activation, parts mating etc.

The Actuation and Sensor Level serves to provide the basic manipulation and sensor capabilities of the robot. The activity at this level is intimately tied to the physics of the domain but no task context is required. Activity occurs in continuous time (practically realized by high rate servo sampling) and is predominantly numeric in nature. Functions at this level are mechanized by the Manipulation and Control Mechanization subsystem (MCM) and the Sensing and Perception subsystem (S&P) with compliant motion execution or object tracking constituting representative examples of subsystem activity.

Similar levels may be identified with the Human Operator, with High Level cognitive and planning functions at the top of the hierarchy, motor skill coordination at the intermediate level and muscular and sensory systems at the lowest level.

Coordination between the autonomous levels of the hierarchy and the Human Operator is provided by three methods. At the lowest level coordination is enabled by providing the Operator with the ability to directly interface with the robot arm and sensors via force reflecting joysticks and stereo camera systems. At the top of the hierarchy the coordination is provided via interactive task planning activity between the Task Planner and the Operator. At the intermediate level the coordination is achieved via a mechanism known as Traded/Shared control described further below.

Traded control refers to the situation where the Operator relinquishes control over process level operations for certain segments of the task sequence execution while retaining it for others. The transition between the operator controlled segments and autonomous program controlled segments is achieved via a 'hand-off' protocol that ensures that information about the world state is correctly communicated between the operator and the machine.

Shared control refers to the situation where the Operator and the autonomous programs jointly participate in decision making at the process level. The contribution of the autonomous programs could be passive in nature as in the form of monitoring operator actions, e.g., ensuring that joystick actions will not lead to possible collision with the environment. The participation may also be much more active with the autonomous programs actually contributing to the manipulation of objects in the environment, e.g., letting the operator control the wrist of the robot arm while the autonomous programs control its global positioning.

The overall structure of such a system architecture is depicted in Figure 1.

3. Run-Time Control Architecture Requirements and Constraints

We now focus on the architecture for the Run-Time controller. The functions of this architecture are to:

1) Mechanize Process Level Operations
2) Permit Man-Machine Coordination

The specific design of this architecture is influenced by the context within which the telerobot operates, the details of which are described below:

1) Man-in-Loop

This feature has the most significant impact on the design of an RTC architecture. The existence of the capability of teleoperation implies that the autonomous capabilities of the autonomous RTC architecture need not be complete. It is sufficient to have an architecture that is capable of performing autonomously for a reasonably high percentage of the times the task is required. Since most computational algorithms to plan and mechanize simple physical subtasks such as grasps etc. are exponentially hard to perform, this feature of the architecture enables algorithms to be selected based on their likelihood of successful performance.

The architecture is also required to functionally interface to both the Operator and the AIP planner. This implies that an Interpreter style command language is required to enable the human to effectively communicate with the system. The Operator must also be able to interact with the system in a manner that does not involve a knowledge of the detailed implementation and computational aspects of the system. This requirement is usually taken to mean the implementation of a Task Oriented Language where actions are implicitly specified in terms of the required task. This concept is implemented in the RTC via a Task Command Language augmented by a constraint specification scheme. The constraint specification language not only allows the operator (or the AIP) to designate the required physical task, but also allows the specification of a set of constraints that are to be maintained by the autonomous system during the execution of such a task. The autonomous system then determines the appropriate actions to execute the task and also satisfy the constraints. A sample entry of the RTC command dictionary is shown in Figure 2. The operator is also required to interact with the system in a shared mode of operation as described earlier. The constraints specification command language allows a natural and easy extension to permit the specification of shared control activity.

2) Specialized Environment

The telerobot is required to operate in a space environment performing tasks such as satellite repair, structure assembly etc. The specialized nature of the tasks implies that standard factory automation paradigms are inapplicable. Further, unlike a highly changing manufacturing environment, the space application environment does have a substantial body of off-line knowledge available in the form of actual astronaut experience, earth simulators etc. The existence of this knowledge indicates that script based techniques are feasible for decision making, because they permit the transfer of the available knowledge to the system with the least effort. Of course, as the telerobot takes over more autonomous functions, the scripts could be replaced with modules capable of reasoning and generating their own activity.

The other constraint of the specialized environment is the perspective shaped by the fact that the Run-Time Controller will eventually form part of an Embedded system implemented in space qualifiable hardware/software. Also, since the controller is part of an end-to-end telerobot system, requirements of overall system design affect the design of the local subsystem architecture. While these constraints are not rigid since the first implementations of the JPL telerobot are necessarily ground based research prototypes, the requirements for an evolutionary growth in the system require that these factors be considered while designing and building the baseline architecture.

3) Parallel Activity Tracks

A requirement of the telerobot architecture is the management of parallel task execution

and the simultaneous use of many physical resources available to the robot. Physical resources consist of various manipulation resources such as robot arms, end effectors, mobile platforms etc., or various sensor resources. Each of these resource could be active simultaneously as in the example of dual arm independent or coordinated actions. The architecture must be embedded with appropriate resource logic to ensure that resource conflicts do not occur when simultaneous activity tracks are in progress. Further, the error detection and recovery for each of these independently controlled resources must be coordinated.

4) Computational Bottlenecks

Computational resources must also be managed. Most robotics algorithms are computationally very intensive, far outstripping the capabilities of uniprocessor machines. Concurrent processing in a distributed multiprocessor environment is necessitated, and the system should therefore be capable of performing the dynamic computation task allocation necessary to optimally use the available computational resources available to the system.

In the RTC, the computational paradigm that has been adopted is the message passing multiprocessing system.

5) Robust Operations

Robust operations at the any level of the hierarchy require the implementation of feedback, where the actual progress of a subtask execution in the world is monitored to guide further actions. At the sensor and servo level, this feedback is implemented via various control system schemes designed to operate with a continuous physical world. The RTC, on the other hand, implements a feedback scheme that operates in event space. A more detailed description of an event space formulation of the RTC decision making is given in the next section.

4. Event Control

The notion of event space and event control is intimately connected with the time horizon of the subsystem of interest. It can be argued that reaction time for a subsystem at any level of the hierarchy should be sufficient to react to a possible world change that is relevant to the goal commanded to that level. Therefore for any subsystem, its time horizon should be comparable with the time constant of the control processes at that level. While the precise determination of appropriate time horizons must necessarily be ad-hoc, it nevertheless allows the definition of a hierarchical control scheme, where a higher level relies on the lower subsystem for the execution of commands associated with small time constants.

A hierarchy in the chain of command is naturally coupled with a hierarchy in the processing of sensory information as well [1]. This leads to an important simplification in the overall task execution monitoring activity in the telerobot. That is, the subsystem at the commanding level is relieved from the mundane supervision of command execution at the lower levels, and is instead free to focus on monitoring and anticipating a smaller (and finite) number of crucial events.

Taken together, a "trusted soldier principle" [2] may be formulated. According to this principle the overall process-level decision space can be factored into two distinct components. One component is acted upon by the RTC, while the other is the sole responsibility of the sensor and servo subsystems.

On any level of hierarchy, Event Control requires the mapping from the space of sensor event sequences:

$$\{S_j^k\},\ i = 1..M_k\ K=1..S_n$$

to the space of actuation event sequences available for this level:

$$\{A_i^k\},\ i = 1..N_k\ K=1..S_n$$

where 'i' is the running index for the sequences and S_n represents the number of lower level actuation/sensor subsystems.

While this formalism can be used to describe a traditional servo system with digital elements in it, its main goal is to create a finite parameterization of actuation and sensor events therefore making error recovery through search possible.

The "trusted soldier" principle leads to a unified command format. Commands to a lower level specify goals, constraints on the allowable ways to reach this goal, specifications on when activity should be stopped, and report formats as well.

In our case the communication between the levels of the event control space (RTC) and the continuum space (MCM and S&P) is performed by a Primitive which is modelled as follows:

Primitive T = (P,M,F,R)

where:

P - desired trajectory (position x force space)
M - desired class of control law
F - desired reflex predicate (stop function)
R - desired event reporting function

The components P and M specify the physical motion to be executed by the robot. The component F determines the termination of the specified physical motion via a sensor predicate which when true triggers the appropriate reflex action (typically a stop). The component R determines the sensor predicates that generate the event reports to the RTC.

The specification of P and M determines the desired map from the actuation event space to the continuum sensor/actuation space. The component R specifies the map from the sensor space to the sensor event space. The component F specifies the precomputed reflex actions that must be taken by the MCM and represent the algebraic (memoryless) and local components of the decision map that are to be executed within the fast decision time constants of the MCM subsystem. We want to emphasize that the stop function F is to be defined before the start of the actual command execution, therefore allowing preplanning for all possible outcomes of T.

Event Control in the RTC

The interface between the RTC and the lower level subsystem adopts the "trusted soldier" principle of operation. That is, the lower level subsystem has the full responsibility of performing the desired command T. If T is unsuccessful then the role of the RTC is solely that of coordinating and interpreting the various reports R (from all of the subsystems) to determine the next command T to be sent to one of the lower level subsystems.

Given this mode of operation, consider the actions to be taken by the RTC on reception of a command from the AIP or the operator. The command may be eventually translated according to one of the four following cases:

1. COMMAND --> (P,M,F,R)

2. COMMAND --> (P_1,M_1,F_1,R_1)
   .
   .
   --> (P_n,M_n,F_n,R_n)

3. COMMAND --> $[(P_1,M_1,F_1,R_1)\ (P_2,M_2,F_2,R_2)....(P_m,M_m,F_m,R_m)]$

4. COMMAND -->

   $[(P_{11},M_{11},F_{11},R_{11})\ (P_{12},M_{12},F_{12},R_{12})....(P_{1m},M_{1m},F_{1m},R_{1m})]$
   .
   .
   $[(P_{n1},M_{n1},F_{n1},R_{n1})\ (P_{n2},M_{n2},F_{n2},R_{n2})....(P_{nm},M_{nm},F_{nm},R_{nm})]$

Case 1 corresponds to a single possible primitive T (albeit parameterized) that can be sent to the lower level subsystem. An example can be a command to move an arm to a given joint position using given joint interpolation.

Case 2 corresponds to the case where one of many primitives T may be sent to the lower level subsystem. A command that specifies end position, but not the specific trajectory to be selected by the RTC, can be used as an example here.

Case 3 corresponds to the case where a sequence of primitives T is necessary for process-level command execution, and case 4 deals with the situation where more than one sequence is available.

Since only one command can be send to a lower level at a time, the ambiguity that exists in the cases 2 and 4 must be removed. There are several possible ways to do it. A "hard rule" approach assumes an existence of an implicit agreement between the commanding subsystem and the RTC on how a single fourplet should be chosen. For example it may be assumed that a straight line interpolation should be always used. In this case there are no real differences between cases 1 and 2. On the other hand a "soft rule" would allow the RTC to rank all candidates and start execution from the "best" one. If it failed, and the action were reversible, then another candidate could be tried. The RTC implements a mixed approach.

It ranks possibilities internally based on an RTC internal criteria; but after a choice is made and execution started, the rule becomes "hard," and an error in execution would require the RTC to report it back to the commanding level without exploiting the rest of the options.

Commands to the RTC which result in Cases 1 and 2 are called Actions. Commands resulting in Cases 3 and 4 are called Skills. Skills require a sequence of event control decision making prior to the issuance of each lower level subsystem primitive T. In Case 3 the sequence of primitives is constrained by the one possible mapping indicated earlier, but the numerical parameterization of the individual T is determined at run-time by the RTC. On the other hand, Case 4 allows the possibility that a different sequence of primitives together with their individual parameterization may be selected during the middle of the execution of a specific sequence. Since the creation and execution of this sequence is dynamically determined by the RTC, the ability to change the parameterization of a primitive T, or the actual change of a sequence, gives the RTC the ability to perform "trimming" (i.e., error recovery of subtask command execution). Note that by virtue of this definition of trimming, Actions cannot be trimmed.

Using an analogy from control theory, the modules that implement the 'forward loop' operations of Action selection or a nominal Skill sequence determination are incorporated into a Process Level Planner, and the 'feedback loop' that analyzes events and determines the need for trimming is incorporated in Monitor, Predictor, Evaluator and Trimmer modules. The feedback paradigm corresponding to this event control system is shown in Figure 3.
It should be noted that two key assumptions govern the functionality and design of these modules. It is assumed that a nominal task sequence is known from off line analysis, and that in a case of ultimate failure, control can always be surrendered to the Operator.

For the MCM a particular implementation of command components P,M and R is given by:

P - a set of via points in joint/task space, or spline functions.
M - position mode or compliant mode control.
R - standard report information returned to the RTC after stop (or reflex actions): joint positions, force/torque sensors readings, and the reason for end of execution.

The choice of the stop functions set F is more complicated. It is clear that F must be complete in the sense that some condition must be eventually triggered. The triggering conditions must also be unambiguous, namely only one stop function should be triggered at a time. Such a choice of the stop functions makes the number of possible outcomes for each MCM command finite.

The process level planning is based on scripts. This allows the embedding of careful off-line analysis of the possible outcomes. In the future an expert system can be implemented to make a choice of an appropriate trimming action. Another advantage of scripts lies in their ability to incorporate ad-hoc domain specific knowledge for planning of a nominal, feed-forward part of a skill. For example an approach position for a grasp can always be determined as a set distance away from the grasping point. Another specific feature of the current implementation is the existence of a command-object cross table which provides for every action, a list of parameters needed for nominal planning of this Action for each known object. It can be filled off-line or, if appropriate modules are available, on-line upon demand. The modular nature of this architecture allows system modification and extension in an orderly fashion.

Skill implementation includes two different phases: planning and execution. First an appropriate script employs a generate, test and modify paradigm to make a nominal sequence of actions. At the same time the script provides a list of possible trimming actions that can be employed to recover from the errors on the intermediate steps. Then the execution is performed according to the following loop:

```
WHILE Action_Stack Not_Empty
loop:
     Bind Action Parameters;
     Set_up Evaluation_context;
     Send Action to the MCM;
     Wait for Report;
     Evaluate Report;
     if SUCCESS then
         null;
     else
         Determine trimming
           if ANY AVAILABLE then
             put trim_action on Action_Stack;
           else
             Report_up(Fail);
           endif;
     endif;
  endloop;
```

An actual example of a script to perform object grasping is shown in Figure 4. Numbers after each action refer to the trimming stack associated with the grasp script.

5. Implementation

The previous sections have described some of the techniques used to perform certain desirable simplifications, such as mapping a continuous problem space into a discrete one. Now the structure and implementation of the RTC architecture will be examined.

As background, the software implementation of the RTC is on an AI VAXstation II, under MicroVMS. The software is written primarily in Ada, with some portions in C and Fortran, which are accessed with the Interface pragma supplied by Ada. Ada was our choice of language for the RTC deliverable software, but we perform rapid simulation and prototyping of algorithms in Prolog, Lisp, Smalltalk and a locally-written development environment called Thread. Our choice of Ada for the deliverables was made for several reasons, the most obvious being that the RTC will eventually have to comply with the Space Station all-Ada software requirement anyway. In addition, the ability to express concurrent programs directly in the language without resorting to operating system interfaces was also a desirable feature. Our strongest reason, however, was the extremely high modularity of Ada, allowing several people to work on the same software without disastrous consequences. This, coupled with the enormously high level of compile-time error checking in Ada, turns out to have been a true expectation. Our software production has been only loosely coordinated among four different people, but it has experienced no incompatibility problems and has a history of phenomenally low levels of runtime errors. We believe that Ada is a good choice for our deliverables since they are of precisely the category of software Ada was designed to fit, namely distributed embedded real-time systems. We would make the comment, however, that Ada is not a good choice of language to prototype in.

The telerobot will be used over the course of several years for many purposes, including as a testbed in which to investigate both low (e.g. feedback control laws) and medium (e.g., grasp strategies) level robotics algorithms, as well as for the targeted demonstration scenarios. For this reason, the architecture of the RTC must have the character more of a development environment rather than of a finished product. The ability to reconfigure the telerobot to perform servicing tasks on satellites other than the one targeted for testing is one of the obvious requirements. It is our opinion, however, that the many complex interactions which can take place during various types of robot operations, even in such a structured situation as the servicing of a modular satellite, would make it likely that even state-of-the-art robotics algorithms would be far too rigid to meet all needs. To insure sufficient flexibility, one must not only provide as much as possible in the way of current robotics algorithms, but also provide the ability to completely change how, when and which algorithms are applied.

The ability to do such general restructuring within a practical robotics implementation has very large software consequences since the addition of a particular algorithm to the mix used in the robot might require brand new data representations to accommodate new information relevant to previously unmodeled properties of objects, and the addition of the new algorithm could very well require completely redesigning the strategies used for applying various currently-used algorithms. An example might be the addition of a new grasp planner module, which might require that more data concerning each object to be grasped be installed in the data base and that the previously used strategy for stable grasp position determination be disabled or modified.

In order to support such a high degree of flexibility, the structure of the RTC architecture is that of a framework in which robotics algorithms can be installed, rather than a particular mix of algorithms. The RTC consists of several modules with a very rough degree of functionality assigned to each type of module, but the specific inputs, outputs and details of operation which each module performs are controlled by reconfigurable data sets supplied to each module. The rough partitioning mentioned above is relatively simple and is an ad-hoc solution to the question of how to slice up the problem of processing the commands received by the RTC. One important characteristic, namely the ability for the RTC to accept commands in parallel, had a major role in selecting the problem separation.

The RTC is required to be able to perform more than one command simultaneously, assuming that they do not conflict with each other. The task of managing access to various sharable, non-shareable and queue-based resources is fairly straightforward in the context of the telerobot, due to the fact that the autonomous system may be interrupted by the operator at any time. This makes it virtually impossible to perform any time-scheduling of resources, so the simple restriction is made that all commands must be independent of each other with respect to time shifts. This removes the necessity of performing the many complex sorts of resource allocation and activity coordination which would otherwise be required between commands. This restriction is not as crippling as it seems, however, since there is no rule which states that a single command can use only one resource. In fact, if a complex cooperative task involving two arms and a vision system is desired, it would simply need to be formulated as a single command so that the coordination operations necessary could take

place within the framework of one command, without requiring propagation to other activities. This independence also greatly simplifies the task of managing response to unexpected error conditions, since all that is necessary is a simple termination of the command, or of all currently executing commands if the error is a global one. Again, this simple behavior is not restrictive, since any sort of complex error response can be produced, if desired, simply by including it as an expected possible off-nominal situation in a command. Obviously, there is a great deal of possible interaction between robot actions in the real world, and formulating two commands in such a manner as to be truly time-shift invariant might be very difficult in any given situation, but this is not really an issue. If the two operations are unconnected, they can simply be done in series, formulated as two separate commands. If they must be done cooperatively, they can be formulated as one command. The performance of dependent operations in parallel could aid efficiency, but is not a key requirement.

With the above preface, the ad-hoc decomposition of the RTC's processing of commands follows.

Command Parsing
- An incoming command from the higher-level system is first converted into some internal working data structure used by the other modules; this allows decoupling internal operations from external interfaces.

Script Elaboration
- The command is then examined, and a specific sequence of activities is decided upon to execute it, together with specific possibilities for responses to off-nominal events during execution. An example of this would be a grasp command, for which a sequence of three straight-line motions followed by a closure of the gripper was decided upon, with the off-nominal behavior of backing up to the starting point and simply repeating the entire motion if the grasp does not succeed on the first try. This sequence is specific in the number and type of motion primitives to be executed, but no details are present yet as to which trajectory is to be used or what the point of grasp on the object is to be.

Action Binding
- An elemental motion/sensing primitive in the sequence is then further processed, in order to determine the precise numerical parameters for its execution. This primitive is then sent to the appropriate subsystem, which responds with one or more reports back to the RTC providing progress/result information about the command's execution.

Report Analysis
- The returned report(s) are examined, and a determination is made whether to continue execution, abort or take some off-nominal corrective action, such as the retry option mentioned above. If the decision is made to continue, the previous step is then repeated for the next primitive in the sequence, followed by this one, until all primitives in the sequence have been executed.

The overall architecture of the RTC is one in which each of the steps outlined above has been assigned to one type of module, with each command thus being processed by one of these four types of modules at any one time. Multiple modules execute in parallel and communicate by message passing in a very simple way. New instances of each module can be created as needed to process parallel commands since each module is simply a worker which is dispatched with a job to do and then returns with a result, with no permanent memory of its own and (with a single exception) no side-effects. All information relevant to the processing and execution of the command is contained in the inputs and outputs to each module, with the exception of spatial/geometric information about the state of the world, which is contained in a global database. This database can be read by any module, but only the module assigned to analyze the reports returned from an executing subsystem has the authority to write to it, thus updating the information in the database with the data returned by the executing subsystem. This is the single side-effect present in the execution of any module, which removes the possibility of write contention and other such coordination problems.

The RTC is thus made up of zero or more copies of each of the four command-processing modules, together with several additional modules which perform various essential functions:

- An Interface Server, which serves as a communication port to the other subsystems in the telerobot.

- A Decision Unit, which is the central dispatcher/coordinator for the RTC. This is simply a finite state machine, with the four-phase command processing behavior built into it, along with the terminate command-on-error and resource-management behaviors. In the nominal case, it simply feeds the input command into successive modules, propagating one module's output to the next module's input, with minor exceptions. In anomalous situations, it simply takes several straightforward steps to insure that the executing subsystems and internal modules working on that command are shut down and sends a report of the halt back to the higher level system commanding the RTC.

Because of the simplicity of the Decision Unit (a simple decision tree), it needs only a very short period of time to execute any given response to an event, such as an incoming message. For most cases, it simply sends out the incoming data to the appropriate module. This allows very rapid response to exception conditions, such as a command from the telerobot's human operator to stop executing. The Decision Unit will almost certainly be idle within a few milliseconds of receiving the message, and can then process it, and send out subsystem halt primitives very rapidly, giving only a few tens of milliseconds of delay between the operator's commanding the halt and the arm actuation subsystem halting arm motion.

• Command Parser modules, as many as needed, which convert the input to an internal working form. It should be explained that one desirable feature of the telerobot is to let the operator intervene in execution at any level of the command hierarchy from AI (activity planning interface) down to detailed arm motions (teleoperation), so the RTC's command inputs are also of human-usable form. The command input from the higher level is thus in the form of an ascii string with fairly readable content. From the RTC viewpoint then, there is no distinction made between the human operator and the AI planner.

• Script Elaborator modules, as many as needed, which decide on the basic script to follow for executing the command. In cases where a parameter for an earlier primitive action must be determined by using a precise value for a parameter to an action to be executed later, (e.g., a precise grasp point may be needed to backtrack a trajectory to the current position of the arm) then determination of these numerical parameters is also performed at this phase, and the detailed parameters are inserted into the primitive before it is sent to the next phase. This permits the use of backtracking as a planning technique if desired, along with any other technique which requires a noncausal determination of specific parameters for actions.

• Action Binder modules, as many as needed, which determine the precise numerical values needed for each subsystem primitive to be sent out for execution. The Action Binder would be invoked once for each primitive in the script, whether the primitive was a nominal action or a response to some planned-for off-nominal condition. The stepwise invocation of the Action Binder allows the use of parameters determined at one point of the execution of the command during run-time to be used at later points in a very simple fashion. Also, as described above, a parameter for an action primitive may be already fixed when it arrives in an Action Binder, which will not disrupt the normal operation of the module.

• Analysis Unit modules, as many as needed, which examine the reports returned by the executing subsystems, update the global geometric/spatial database accordingly, and determine the recommended course of action to be followed, which would be to continue normally, abort, or take a specific foreseen corrective action. This recommendation is then sent to the Decision Unit, which may choose to follow it, if overall execution is normal, or may ignore it and choose to terminate the command, if for example the AI has sent a halt instruction to the RTC during the execution of the command. The key point concerning the AU modules is that they operate entirely on command-related information, and do not take into account such things as global anomalies, but operate as a memoryless single-input single-output system. The input is the command context together with the report sent back by the subsystem, and the output is the recommendation for action. The Decision Unit worries about coordinating any overall behavior which crosses command boundaries.

• A Database Server module, which simply behaves as a shared-memory area for the other modules. Any given location in it can be written to by only one AU at any given time, and there are various validity flags to indicate whether or not other modules should be allowed to read any given piece of information.

• In addition to these modules, which form the basic configuration of the RTC as shown in Figure 5, there are several others which will not be detailed, but which exist because of the requirement that the RTC's geometric database be accessible to any subsystem in the telerobot, to act as a central repository. Also, there is the requirement that the AI planner, in order to perform its planning, may need to use backtracking methods. To support this, the RTC contains an identical copy of the Command Parser/Script Elaborator/Action Binder modules which are used as a hypothesis-testing facility by the AI planner. The planner can simulate as much as possible of the execution of a possible RTC command, without actually commanding any subsystem activity, so as to determine whether or not a particular line of planning will be found to be infeasible at the numeric level by the RTC. This facility allows the fairly ad-hoc division in the planning made between the AI planner and the RTC to function robustly in the presence of couplings between the symbolic and numeric levels of robot activities.

The primary aspect of the modules which perform command processing is that they are implemented not as hardwired entities which operate on data, but as data-driven "interpreters," which take as input a program, as well as data. This implementation allows the modules to be easily reconfigured to use a new algorithm if at all possible. For example, adding a new algorithm may absolutely require a complete restructuring of the existing system because of robotics considerations, but such restructuring would not be necessary simply because of software difficulties. This feature allows the addition of many types of robotics and also AI techniques to the RTC if desired since there is no restriction at all on what data is passed between the modules, and only trivial restrictions on the order in which they are invoked. It is our hope that later versions of the Script Elaborator will use AI-type reasoning techniques to produce the basic scripts for use during execution, rather than the cut-and-paste/table lookup technique used now. This would give the ability to literally replan a sequence in the event of an error, instead of forcing error responses into specific scripts. Also, a prediction capability, capable of utilizing expectation information in analyzing the sensor data during execution, would be a desirable feature to add to the Analysis Unit. The ability to remove the independency restriction on separate commands and perform coordination between more than one command would also be useful in increasing system efficiency. All of these ideas have been explored in a preliminary fashion, and several fairly straightforward implementation alternatives have been found to each of them, indicating that the ability to specify module behavior at run time, rather than just its input data, is an extremely powerful feature of the RTC.

A feature of the RTC is the fact that the memoryless, dispatched-worker module format is ideal for implementation on a multiprocessor hardware architecture. Modules do not perform any significant nonlocal references (the exception is database access, which can be reduced by simply grouping read requests into bunches, rather than lots of individual read requests) and do not require any communication among themselves during execution. A preliminary study indicates that conversion of the RTC to operate on a hypercube multiprocessor would be a very straightforward task.

Another aspect of the RTC's internal operation is that it is relatively simple to treat the human operator as a subsystem to be commanded by the RTC. This allows several simple but effective solutions to the extremely difficult problems of specifying to the autonomous system what the intentions and outcomes of human teleoperation activity are. For example, if the operator intervenes into autonomous execution and picks up an object, there is no way for the autonomous system to examine arm trajectories and gripper force data to determine that the object was grasped at all, or if it was grasped what the position of it in the gripper is. This very simple example shows how difficult is the task of sharing control of an autonomous system with a human operator. One solution, which by necessity imposes a good deal of overhead and restriction on the human operator, is to specify intervention activities to the telerobot in the same form as any other command, with the exception that the performing subsystem is the operator. This paradigm would mean that precise numerical parameters would be left out of each step, but the normal sequence of subsystem primitives would be used by the RTC, and likewise the usual sequence of RTC commands would be sent out by the AI, and the operator would face the restriction of performing only that portion of the task specified by the primitive. An exploration of this method of structured operator intervention, which is one candidate for the JPL telerobot, is given below.

If an object could not be reliably grasped by the autonomous system, the operator could instruct the AI planner to

"grasp Object with Right_arm via Operator_intervention."

This would result in the command trickling down the hierarchy through the RTC and a subsystem primitive to the operator appearing on the control console. In this example, the first might be:

"move Right_arm to_neighborhood_of Object".

The operator would perform this, in teleoperation mode, and indicate that he was finished. The RTC would then send out the next primitive:

"move Right_arm to_grasp_point_of Object".

The operator would comply, seating the gripper on the object in a satisfactory configuration so that when the gripper was closed, it would grasp the object firmly, <u>without moving the object</u>. The RTC would then be able, by performing kinematics and geometric computations, to know what the grasp point reference frame for the object was and would, therefore, be able to correctly update the data base as to the position of the object after it has been grasped. The final primitive would be sent by the RTC, directing the operator to close the gripper, and the operator's response would confirm that the grasp took place as expected, without disturbing the spatial relationship set up.

This scenario demonstrates the additional overhead imposed on the operator by the necessity of maintaining the autonomous system's integrity. It is essential, however, that some form

of strong restriction be placed on the operator, not only to coordinate activity with the autonomous system, but to prevent operator disruption of the autonomous system due to human oversight. It is likely that there will need to be more effort put into protecting the autonomous system from the operator than into making the autonomous system operate effectively on its own.

Another issue we believe to be of foremost importance in the telerobot design is that of detection of anomalous conditions in the world. Even in such a highly identified environment as satellite servicing, it is crucial that sensor feedback be employed as often as possible so as to prevent any cascade of errors forward through the execution of a servicing task. It is unlikely that any AI/robotics autonomous system will be able to make allowances for error propagation in its activities, within the near future. Such propagation must be eliminated if the autonomous system is not to become completely confused, with major portions of its world model invalid, which is a completely unacceptable situation owing to the large amount of time and effort which would likely be required in order to restore the world model to a correct state (work weeks).

The development of such an architecture must necessarily recognize the limitations of current science and technology in this nascent area. Early architectures focus on integrating the process-level autonomous functions into the system for simple, independently controlled arms. Operator-machine coordination is restricted to simple traded control schemes. Later architectures support more complex physical environments (more arms, redundant arms), as well as more complicated functionality such as coordinated arm motion and true shared control. Concomitant with this increased complexity and functionality is the management of complex computational architectures and the integration of more sophisticated error recovery and planning methods into the system. At the present time, the RTC of the JPL Telerobot has been implemented to contain the following capabilities:

- Command Parser -- parses ascii strings from a simple BNF form into a record data structure containing equivalent information. The BNF language in use has roughly 100 terminal symbols, 50 clauses.

- Script Elaborator -- uses a simple decision tree to splice portions of command sequences together into a coherent script, together with off-nominal scripts. Has the ability to backtrack from goal point for purposes of trajectory generation.

- Action Binder -- performs generation of joint/task space linearly interpolated trajectories in a piecewise fashion, using kinematic and load-carrying constraints of the manipulator arms, together with collision detection, to plan small arm motions for grasping purposes.

- Analysis Unit -- uses a table-driven decision tree to evaluate the outcome of an execution and makes a recommendation. If the recommendation requires corrective activities, then the script to perform it is simply looked up in a table, having been generated by the Script Elaborator.

6. Conclusion

We have described a Run-Time Control architecture for the JPL Telerobot and discussed associated issues. This work was performed at the California Institute of Technology, Jet Propulsion Laboratory under a contract from the National Aeronautics and Space Administration.

7. References

[1] J.S.Albus et al., "Hierarchical Control for Robot in Automated Factory." Proc. of 13th. International Symposium on Industrial Robots, Chicago, 1983, pp. 29-43.

[2] A.Lokshin, K.Kreutz "Towards a Hierarchical Robot Control Language", IEEE CS Workshop on Languages of Automation. August, 1986, Singapore.

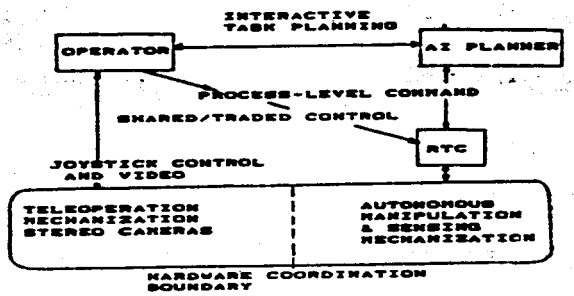

Figure 1. Telerobot Functional Architecture

```
grasp --> GRASP goal maintain reflex
     goal --> end-effector graspable-object grasp-point
          end-effector --> LEFT-END-EFF
                       --> RIGHT-END-EFF

          maintain --> nil
                   --> MAINTAIN safe-work-range
                               avoid-task-region
                               avoid-neighbors
                               pose-during-motion
                               speed.

          reflex --> nil
                 --> GUARDED
                 --> reflex-items

              reflex-items --> nil
                           --> reflex-item reflex-items
              reflex-item --> UNLESS condition THEN action
```

Figure 2. Sample RTC Command

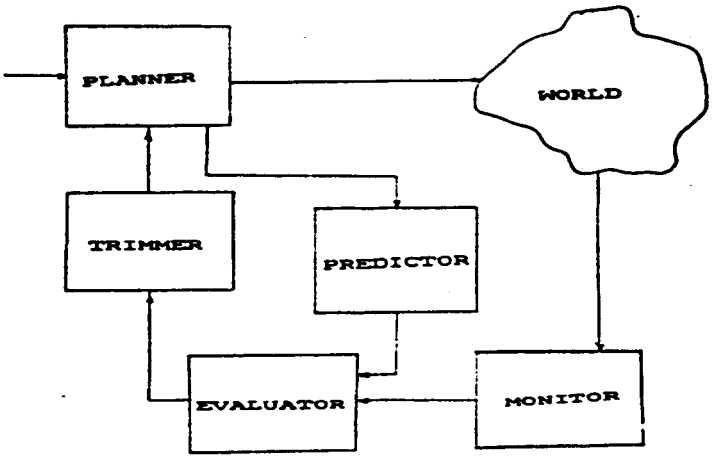

Figure 3. RTC Functional Paradigm

```
SCRIPT

        "grasp_ini"
        0 0
        "grasp_plan"
        0 0
        "active_arm JAP move_free movej"
        0 0
        "active_arm JGW move_to_contact move_joint"
        0 0
        "active_arm GP1 move_to_contact move_task"
        1 2
        "active_arm GP2 move_to_contact move_task"
        1 2
        "GP = 0.5*(GP1 + GP2)"
        0 0
        "active_arm GP move_tocontact move_task"
        0 0
        "close"
        0 0

TRIM STACK

        1       "active_object vision"
        2       "active_object grasp"
```

Figure 4. Grasp Script

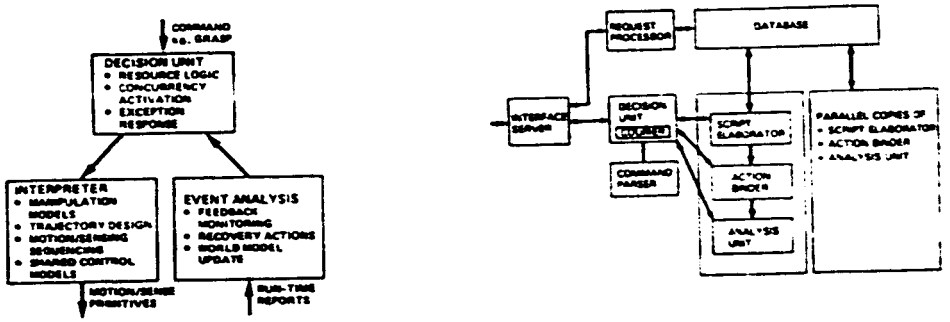

Figure 5. RTC Architecture