NASA-TM-108112

IN-61-TM
15328
P-20

Increasingly Automated Procedure Acquistion In Dynamic Systems

NATHALIE MATHE
SMADAR KEDAR
ARTIFICIAL INTELLIGENCE RESEARCH BRANCH
MS 269-2
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035-1000

(NASA-TM-108112) INCREASINGLY AUTOMATED PROCEDURE ACQUISITION IN DYNAMIC SYSTEMS (NASA) 20 p

N93-16684

Unclas

G3/61 0135328

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA-92-23

June, 1992

Draft paper submitted to the 7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, to be held in Banff, Canada, October 11-16, 1992.

Increasingly Automated Procedure Acquisition In Dynamic Systems

Nathalie Mathé[1] & Smadar T. Kedar[2]

NASA-Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035, USA
mathe@ptolemy.arc.nasa.gov
kedar@ptolemy.arc.nasa.gov

ABSTRACT

Procedures are widely used by operators for controlling complex dynamic systems. Currently, most development of such procedures is done manually, consuming a large amount of paper, time and manpower in the process. While automated knowledge acquisition is an active field of research, not much attention has been paid to the problem of computer-assisted acquisition and refinement of complex procedures for dynamic systems. This paper presents the Procedure Acquisition for Reactive Control Assistant (PARC), which is designed to assist users in more systematically and automatically encoding and refining complex procedures. PARC is able to elicit knowledge interactively from the user during operation of the dynamic system. We categorize procedure refinement into two stages : *diagnosis* - diagnose the failure and choose a repair - and *repair* - plan and perform the repair. The basic approach taken in PARC is to assist the user in all steps of this process by providing increased levels of assistance with layered tools. We illustrate the operation of PARC in refining procedures for the control of a robot arm.

1. INTRODUCTION

Procedures are widely used by operators for controlling complex dynamic systems such as airplanes, nuclear power plants, and the Space Shuttle. Procedures specify how to conduct a set of predetermined sub-tasks (or actions) that are components of a higher level task. For example, they dictate the manner by which a pilot is expected to interact with an aircraft's systems. Currently, most development of such procedures is done manually, consuming a large amount of paper, time and manpower in the process (Mathé, 1990b; Degani *et al.*, 1991). First, a system designer encodes nominal procedures which may be incomplete and incorrect. Then, during the course of system testing and operation, if the procedures fail to perform as expected, the user needs to cope with the failure and find ways to recover. Detecting and correcting the procedures deemed responsible for the failure is done later, off-line. Such procedure refinement never ceases -- it is done incrementally over the lifetime of the dynamic system.

---

[1]Dr. Mathé is a European Space Agency fellow and currently a NASA-Ames Research Associate.

[2]Dr. Kedar is a contractor at Sterling Software, Inc., and is currently a visiting researcher at the Institute for the Learning Sciences, Northwestern University.

There is a need for a tool to assist users in encoding procedures, and refining them during operations. While automated knowledge acquisition is an active field of research, not much attention has been paid to the problem of acquiring and refining complex procedures for dynamic systems. Most attempts at procedure acquisition to date have proposed advanced editors, including flow-chart diagram editors, or other similar graphical editors (Saito *et al.*, 1991). Other attempts have dealt with procedure acquisition in static domains such as medical diagnosis (Musen *et al.*, 1987). While those previous studies mainly addressed the elicitation phase of procedure acquisition[3], we focus in this work on the refinement phase. Research in machine learning addresses procedure elicitation and refinement by providing fully automated methods (Gil, 1991; Kedar, 1991). However, these methods rely on strong assumptions about the domain (e.g. complete perception and prediction), which are violated in most real-world environments.

The Procedure Acquisition for Reactive Control assistant (PARC) is designed to assist users in more systematically and automatically encoding and refining complex procedures for dynamic systems. Thus it is expected to increase the accuracy of procedures with decreased time and effort on the part of the user. In order to be effective, we require that it assist in all stages of the procedure refinement process.

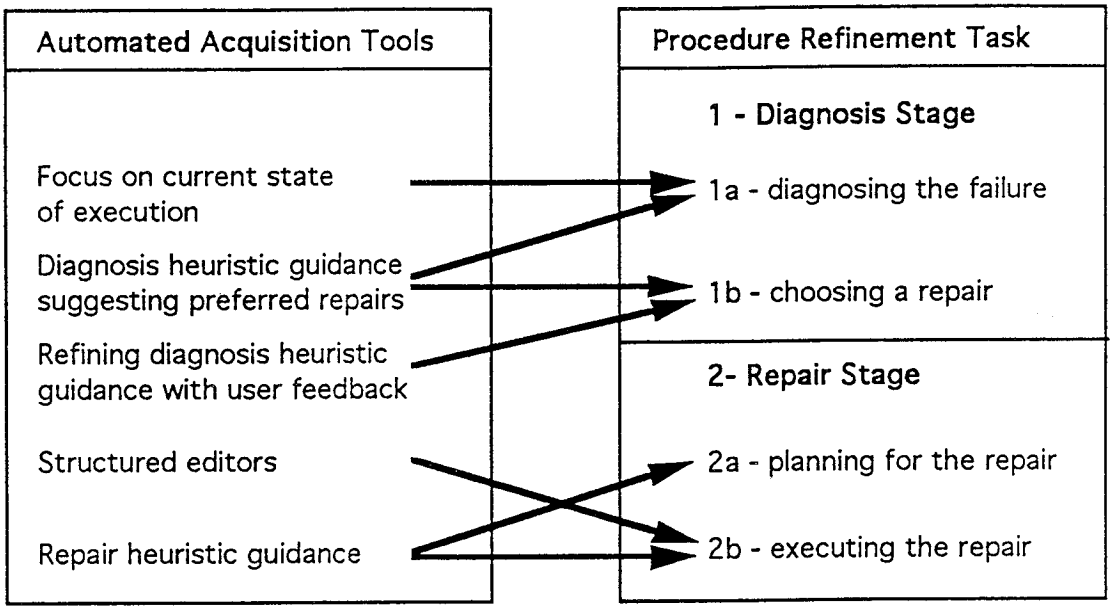


Figure 1: Assistance levels supporting the procedure refinement stages.

Given the experimental nature of procedure refinement, PARC elicits knowledge during operation of the dynamic system. During execution, the user supervises the dynamic system using procedures suggested by a control assistant. When the user disagrees with the control assistant (such an event is called a *failure*), PARC starts a procedure acquisition dialog. We categorize the procedure refinement process into two stages:

(1) **Diagnosis stage:** (1a) the user diagnoses the failure
and (1b) chooses the repair needed to correct the procedure base;

(2) **Repair stage:** (2a) the user plans for the repair
and (2b) performs the repair.

This refinement process is incremental and continues through further execution.

---

[3]Knowledge Acquisition can be broadly classified into three phases: elicitation, refinement and reformulation (Bareiss *et. al.*, 1989).

PARC is designed with layered KA tools, providing increased assistance to automate more steps of the refinement task (Figure 1). At the lowest level of assistance, PARC provides information on the control assistant's current state of execution, assisting the user in diagnosing the failure. It also provides structured editors that can be used stand-alone to perform the repair. Secondly, PARC provides automatic heuristic guidance in both stages. In particular, it suggests which types of refinements are likely to correct which types of failures, assisting both steps of the diagnosis stage. It also leads the user through the repair stage by providing heuristic guidance in planning and executing the repair. Finally, PARC provides automatic reinforcement of the heuristic diagnosis guidance. It reinforces the heuristics with user feedback in order to refine and improve its guidance over time.

In the next section, we will present an overview of the procedure acquisition assistant, including a brief procedure acquisition example, descriptions of the performance system and of the procedure acquisition assistant, and of their associated knowledge representation for procedures. In the third and fourth sections, we describe the procedure acquisition assistant with a detailed running example. We analyze the procedure acquisition assistant in the fifth section and conclude by presenting some related work and future directions of research.

2. OVERVIEW OF THE PROCEDURE ACQUISITION ASSISTANT

2.1. A Sample Procedure Acquisition Dialog

In this section we illustrate PARC's operation using a simple example from a robotics domain. in our robotics lab we use a Puma 560 robot has seven degrees of freedom, a gripper, and several sensors. A sonar sensor measures the distance to the closest object in the environment. A force-torque sensor is used to detect obstacles and perform several types of commands. A camera attached on the gripper is used with image analysis software to detect cubes and compute their positions in the environment. The user has no direct view of the environment. A video screen displays a global view of the robot environment from a fixed camera. The user controls the robot through a graphical interface which includes graphical sensors values displays, a 3D graphical simulator, and a high level command language with menus and graphical command panels (Figure 2). Messages and procedures are also displayed on the interface by the control assistant.

In this example, the task of the user is to build a structure of cubes and beams with the robotic arm. The control assistant suggests executing the sub-task "Approach Cube" using a guarded move robot command. A guarded move uses the force-torque sensor to detect any collision during the motion. The motion is stopped when the goal position is reached (successful command execution) or when an obstacle is detected (command execution failure). The user agrees to execute the "Approach Cube" procedure and observes the control assistant perform the guarded move. Part way through the guarded move, the assistant abandons it to select an "Avoid Obstacle" procedure, trying to avoid a nonexistent obstacle. At this point, the user notifies the assistant that she disagrees; she did not expect the "Avoid Obstacle" procedure in the midst of the "Approach Cube" procedure, because she doesn't see any obstacle on the video screen.

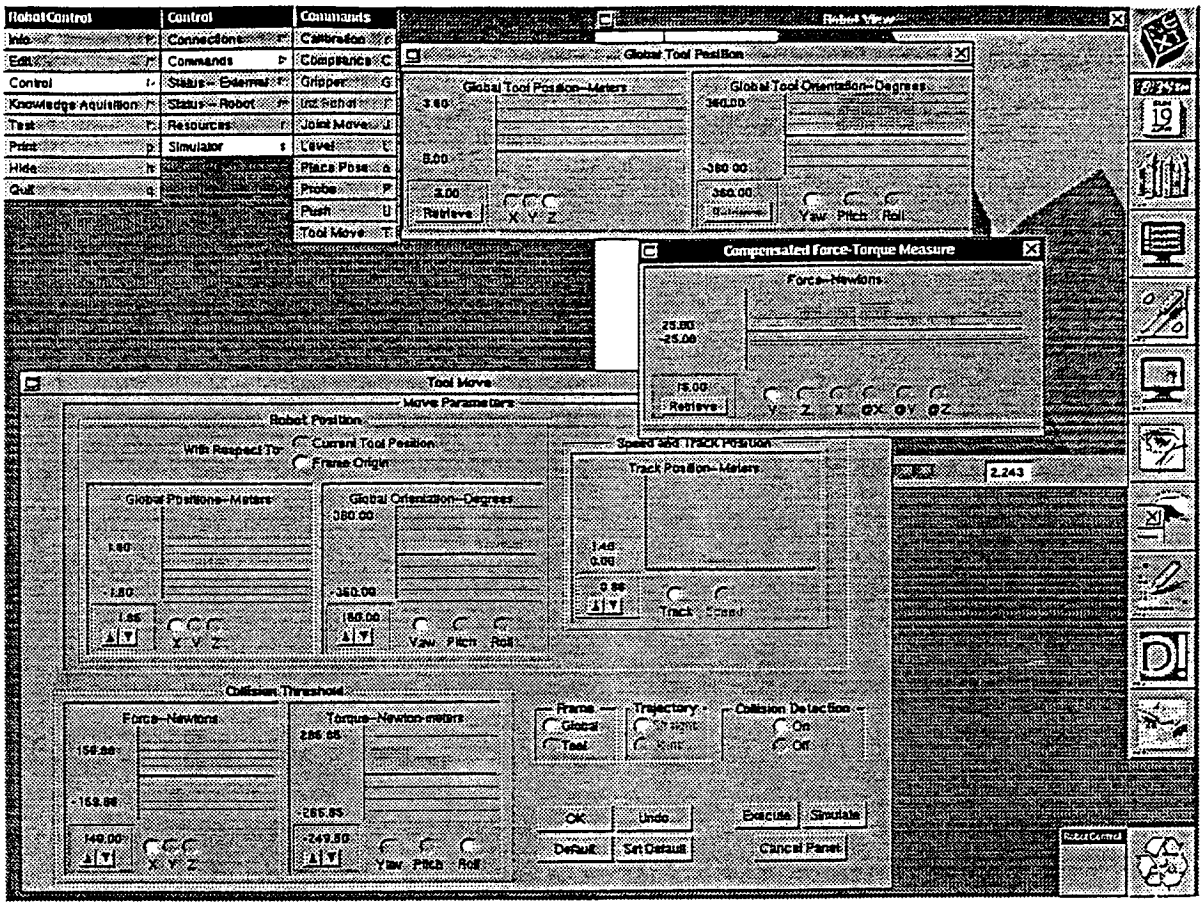


Figure 2: User Interface for Robotic Control.

Given this failure[4], a procedure refinement dialog begins. First, the procedure acquisition assistant provides to the user the system's current state of execution: the procedure for "Approach Cube" is followed by the procedure "Avoid Obstacle" under the abnormal outcome "obstacle". PARC assists the user in diagnosing the failure: there is only one abnormal outcome for the "Approach Cube" procedure, such that any execution failure feedback would be assumed to be an obstacle, and thus to be avoided by invoking the "Avoid Obstacle" procedure. In fact, one of the other reasons for the execution failure of the guarded move command is if the force-torque sensor stops working. The event corresponding to a sensor breakdown has been displayed to the user, and by checking the sensors status, the user discovers that this is the problem. PARC then assists the user is in choosing an appropriate repair for the procedure. Given all possible repairs (adding or removing any preconditions, outcomes, or procedure links), the heuristic guidance suggests that for this type of failure, an additional abnormal outcome and associated recovery procedure is needed. The user selects this repair and PARC then presents a dialog which leads her through the steps of the repair. Later on in system operation, the user finds that once again, when the force torque sensor was not working, the control assistant suggests the "Avoid Obstacle" procedure. By diagnosing the failure, the user notes that the heuristic guidance was incomplete, and that the criticality order of the two abnormal outcomes needed to be modified as well The user's feedback on the repair choice is used to modify and reinforce the corrected heuristic.

[4]Note that a command execution failure is different from what we call a failure: the fact that the user disagrees with the control assistant. A failure is not always associated to an unexpected command execution failure, like in this example.

ORIGINAL PAGE IS
OF POOR QUALITY

2.2. The Architecture

The architecture consists of two main components: the *performance system* and the *procedure acquisition system, PARC* (Figure 3). The task of the performance system is to assist the user in controlling a dynamic system (e.g. a robot, an airplane). It is thus called a control assistant. In our case, the *task domain* is robotic arm control. The procedure acquisition dialog is triggered when the user disagrees with the control assistant. The procedure acquisition system uses the dynamic problem-solving ability of the performance system to support procedure refinement. PARC operates as follows : given the current procedure base and set of active procedures, it assists the user in (i) diagnosing the failure; and (ii) repairing the failure. Once the failure has been repaired (or if the user agrees with the control assistant), the control assistant resumes execution. In the following sections we will briefly describe each of these aspects

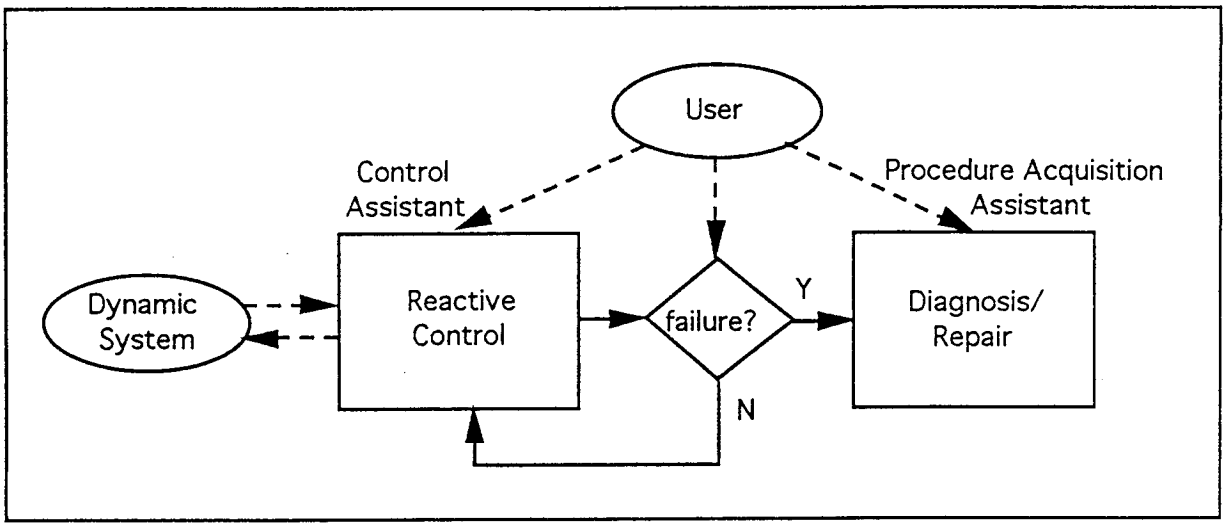


Figure 3: Architecture of the Procedure Acquisition Assistant.

2.2.1. The Performance System

For assisting the user in controlling the robotic arm, the control assistant's *problem solving method* is reactive control: given the task goal and the set of active procedures, analyze the current situation, select the next appropriate procedure, execute it and monitor its results (Mathé, 1990a). Procedures are represented as *Blocks*, a formalism described in section 2.2.3., and the control assistant's problem solving method is based on the inference mechanism associated with the block representation (Boy, 1989; Mathé, 1990b-c, Mathé & Boy, 1992).

The control assistant operates as follows. It monitors the environment status, the robotic arm status and user's actions on the interface. When an event is perceived, it is interpreted by the assistant with respect to its current expectations. The assistant then decides if the current procedure still applies or if another procedure should be selected. In the latter case, it displays a message to the user explaining the new situation and the name of the selected procedure. If the user agrees with the assistant's selection, the selected procedure is displayed on the user interface (by selecting the appropriate display and setting parameters values - Figure 2). This is possible since procedures are expressed in terms of user's actions on the interface (like flight procedures are expressed in terms of pilot's actions in the cockpit), and not directly in terms of robotic instructions common for most autonomous control systems. The user may then choose to execute this procedure, modify some parameters values before executing it, or ignore it and select another command panel. The control assistant then executes the procedure by translating it into low level instructions for the robotic arm and monitoring its results.

2.2.2. The Procedure Acquisition System

PARC *acquires knowledge* that is incorporated as extensions and refinements to an initial skeletal procedure base. This initial procedure base, as well as models of the dynamic system and its environment, is acquired manually during a preliminary elicitation phase with the domain designers. PARC assists the user in incrementally refining this procedure base.

PARC provides tools automating each step of the refinement process. Once the user detects a failure, it assists her in diagnosing why the control assistant suggested the wrong procedure, by displaying information on the set of active procedures involved in the current execution state. Then, PARC assists the user in choosing the repair needed to correct the faulty procedures by suggesting a subset of preferred repairs among all possible repair options, based on some heuristic knowledge. This heuristic guidance can be automatically tailored with user feedback in order to reinforce particular preferred repairs, suggest new repairs, or include domain-specific knowledge. Finally, for assisting the user in planning and performing the repair, PARC proposes a procedure acquisition dialog driven by some heuristic procedural knowledge. This includes knowledge on how to decompose a particular repair into a sequence of steps, on how to take into account the current execution state, and on how to appropriately use the structured editors. Those structured editors provide direct access to procedures and are based on the knowledge representation for procedures described below.

2.2.3. Procedure Representation

Procedures in the control assistant are represented as *Blocks*, a representation designed to facilitate incremental procedure development. In a previous robotic application where procedures were manually acquired through experimental protocols, we developed a cognitive model of reactive control. Following this study, we designed the block representation and its associated inference mechanism as an appropriate formalism for this model (Mathé, 1990b-c, Mathé & Boy, 1992). The basic structure of this formalism is a *block* of knowledge. A procedure base is represented as a network of blocks organized in several hierarchical levels. The inference mechanism associated with this formalism is independent of the content of the procedure base. Different procedure bases can be loaded into the system, depending on the particular application task involved. A block includes the following components:

(1) name;
(2) hierarchical level;
(3) list of preconditions;
(4) list of actions;
(4) list of goals with their lists of associated blocks; and
(5) list of abnormal conditions with their lists of associated blocks.

A block is graphically represented as in Figure 4. It is divided into four parts corresponding to its preconditions, actions, goals and abnormal conditions. The arrows represent the links towards the blocks associated to goals and abnormal conditions. Blocks of lower hierarchical level have a grey background, and block of same or higher hierarchical level have a white background.

Actions are either elementary actions (executable on the user interface) or sub-blocks of lower hierarchical level. Goals and abnormal conditions are terminal conditions which are tested during and after the execution of the block actions. For a given block, each goal and abnormal condition is linked to a list of blocks which contains possible procedures to be executed next if the goal or abnormal condition is satisfied. Abnormal conditions and goals differ only semantically: goals correspond to a successful execution; abnormal conditions correspond to all the abnormal situations which may occur during or after execution. Their associated blocks represent how to recover from a particular abnormal situation.

Depending on its complexity, a procedure may be represented by a single block, a hierarchy of blocks or a network of blocks. A simple procedure is represented by a single block whose actions are elementary. A more complex procedure may be represented by a block whose actions are sub-blocks at a lower hierarchical level, containing themselves either elementary actions or lower level sub-blocks. Finally, a set of complex procedures is represented by a network of blocks, hierarchically organized and transversely connected through their goals and abnormal conditions. Given the procedure hierarchy, the

context of a block is defined as the set of its parent blocks' preconditions. The context structure and hence the hierarchical decomposition of blocks is determined by task analysis. This representation supports incremental procedure development through acquisition of new blocks and refinement of existing blocks by adding abnormal conditions and associated recovery procedures and by learning contextual conditions (Boy, 1989).

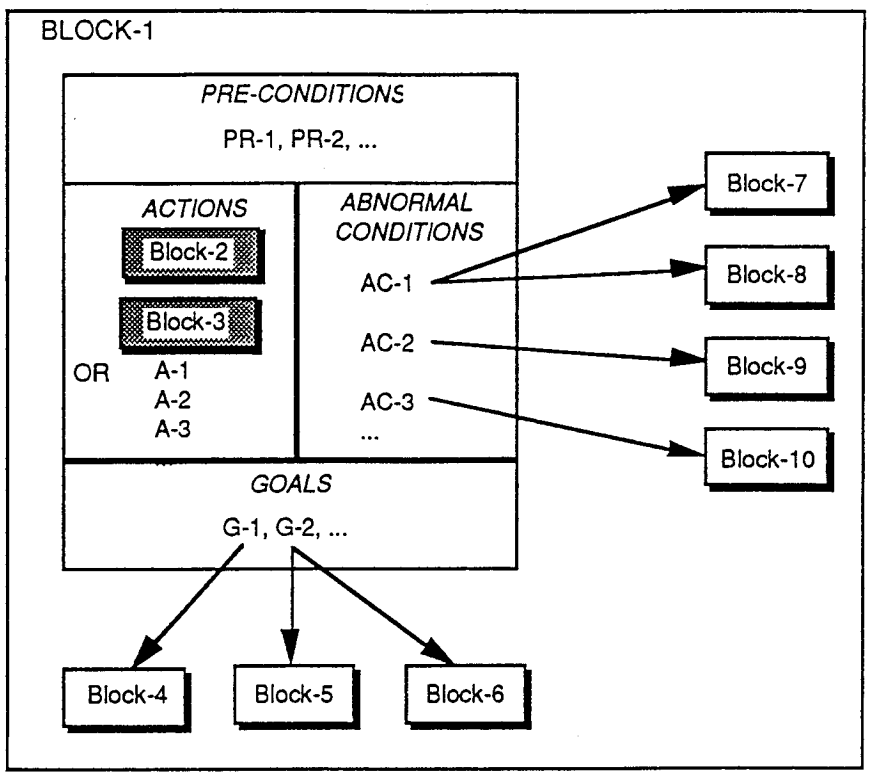


Figure 4: Block graphical representation.

The behavior of the control assistant is based on the inference mechanism associated with the block representation:

(1) Sensory inputs from the environment, the robot and the user interface are matched with the system current expectations. Expectations correspond to preconditions of blocks selected for execution, and to goals and abnormal conditions of active blocks.
(2) The most critical of the best matched expectations is selected by the assistant.
(3) Depending on the type of the selected expectation, the block to which it belongs and the next blocks associated to it, the assistant determines which procedures become active, inactive or selected for execution. If a low level block becomes active, its actions are executed on the user interface. The set of of current expectations is updated and used during the next pattern-matching phase.

2.2.4. Example of Procedures for Robotic Control

Using this representation, we have encoded in the control assistant the first set of nominal procedures corresponding to the cubes and beams assembly task. We have manually acquired this set of procedures through interviews with our robotics expert. For example, the task "Insert Cube in Structure at Destination Pose" is decomposed into several phases: "Choose Cube," "Pick up Cube," "Move Cube to Destination Pose," and "Fasten Cube into Position" which represent a nominal sequence of procedures linked through their goals. The sub-phase "Approach Cube" (Figure 5), part of "Pick up Cube," has one goal "gripper at position above cube" linked to the next nominal procedure "Align Gripper," and one abnormal conditions "obstacle" linked to the next alternative recovery procedures "Find Collision free Path to Goal Position" and "Avoid Obstacle".

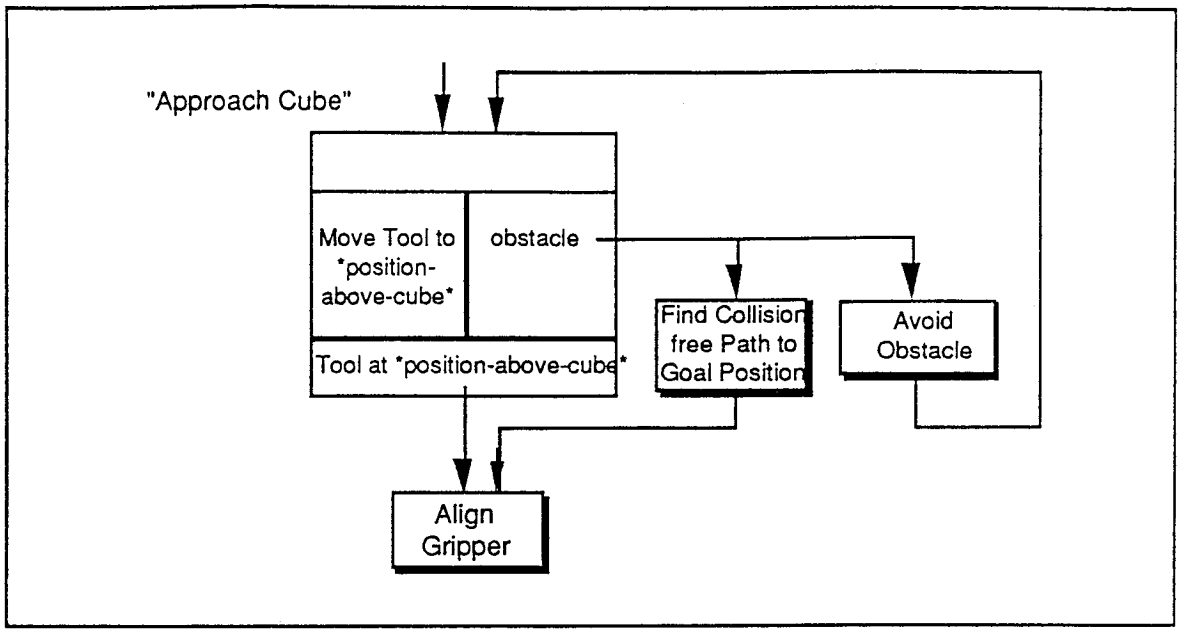


Figure 5: Part of the "Pick up Cube" procedure.

3. AUTOMATING THE DIAGNOSIS STAGE

The procedure acquisition dialog is triggered when the user disagrees with the control assistant. We called such an event a failure. In the following sections we describe the forms of assistance provided to the user in diagnosing the failure.

3.1. Detecting a Failure

During reactive control, the control assistant analyzes the situation, matching it with its current expectations (preconditions, goals, and abnormal conditions). It then decides if the current procedures still apply or if another procedure should be suggested to the user. In the latter case, it displays a message, as shown in Figure 6, informing the user about: (1) perceived events (if any); (2) the recognized expected condition; and (3) the suggested procedure or the result of a previously active procedure (normal or abnormal). The user may agree on the recognized situation and the suggested procedure, in which case the execution will continue (cf. Section 2). Or the user may disagree and start a procedure acquisition dialog, by accessing the other choices considered by the control assistant.

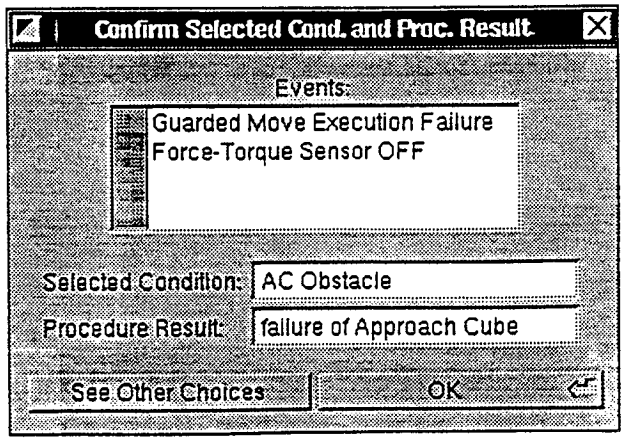


Figure 6: Confirm Selection acquisition panel.

Taking the example described in Section 2, the user wants the robotic arm to insert the next cube into the partially built structure. In order to do so, the control assistant has selected a cube on the table using the "Choose Cube" procedure, then proposed to the user that she executes the "Approach Cube" procedure. This procedure uses a guarded move robot command. The user agrees on this procedure and observes the robot perform the guarded move. Part way through the guarded move, the control assistant informs the user that the "Approach Cube" procedure has failed (Figure 6). The events "execution failure" and "force-torque sensor off" have been received by the control assistant and the abnormal condition "obstacle" recognized. Since the user doesn't see any obstacle in front of the robot on the video screen, she notifies the assistant that she disagrees on the recognized situation, thus detecting a failure.

3.2. Diagnosing the Failure

Once the user disagrees with the control assistant, the first step of the refinement process is to diagnose the failure. The diagnosis consists in determining what is wrong in the suggestion of the control assistant, given the current situation in the environment. For example, no procedure may have been selected if it is missing in the procedure base; or several expectations may have been recognized, but the wrong one selected.

To help the user diagnose the failure, PARC provides information on the assistant's current state of execution. It displays the Situation Description panel (Figure 7) which focuses on portions of the procedure base that are involved in the current execution. This panel describes the current state of execution in a scrollable tree format: current active procedures, current expectations, and next procedures linked to these expectations. The first column displays the names of active procedures in hierarchical order. The second column displays current expectations associated to each active procedure, with their name, type (precondition, goal or abnormal condition), degree of match and degree of criticality. The third column displays the names of procedures to be executed next, associated with each expected condition. It shows the user which procedures may become active next depending on the condition she selects. By default, the condition selected by the control assistant is highlighted when the panel is displayed.

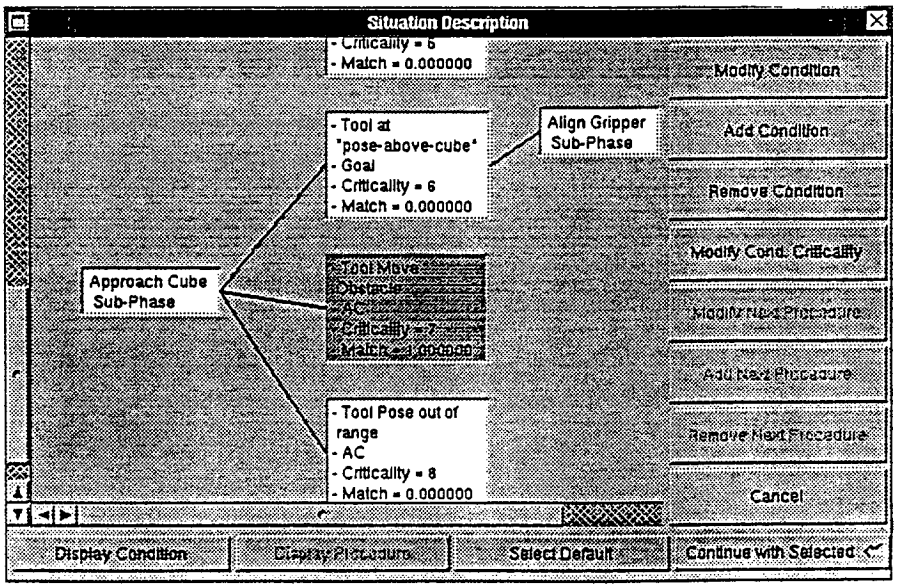


Figure 7: Situation Description acquisition panel.

In our example, PARC displays the set of active procedures in a scrollable tree format: "Build Structure," "Insert Cube in Structure at Destination Pose," "Pick up Cube," and "Approach Cube" at the lowest hierarchical level. Figure 7 shows the bottom part of this tree, including the "Approach Cube" procedure in the first column. In the second column, the expected condition "obstacle" selected by the control assistant is highlighted. It is an abnormal condition of the

ORIGINAL PAGE IS
OF POOR QUALITY

"Approach Cube" procedure. The procedure "Avoid Obstacle" is going to be executed if the user does not change the selection and repair the procedure base. By accessing the "obstacle" condition, the user sees that the "execution failure" event can only be interpreted as an obstacle by the system. In fact, given her knowledge of the domain, she knows that one of the other reasons for the execution failure of the guarded move command is if the force-torque sensor stops working. When selecting the "obstacle" condition, the control assistant has displayed to the user the event corresponding to the sensor breakdown. By checking the sensors status, she decides that this is the problem. Thus, the user diagnoses that the "obstacle" condition has been incorrectly selected and that the sensor breakdown event should have be recognized instead.

3.3. Choosing a Repair

After diagnosing the failure, the user may select a repair or directly access the editors. In the latter case, she has to rely on her own knowledge for choosing, planning and performing the repair. In this section, we describe the assistance provided by PARC for helping the user choose the repair.

First, the set of all possible repair options is displayed on the Situation Description panel (list of buttons on the right hand side - Figure 7): modify, add or remove an expected condition; modify an expected condition criticality; modify, add or remove a next procedure[5]. Those options are not exhaustive, but found to be the most useful for recovering from a failure.

Failures Types		Possible Repairs
No Match	*No procedure associated to selected goal or abnormal condition* 1	-> add next procedure to selected goal or abnormal condition
	No Surprise 2	-> add expected condition -> modify expected condition -> add next procedure
	Surprise 3	-> add expected condition -> modify expected condition
One Match	*General* 4	-> modify the selected expected condition -> add expected condition
	If Abnormal Condition selected and there is no uninterpreted event 5	-> add a new abnormal condition and associated recovery procedure and modify the one selected -> add a new abnormal condition and associated recovery procedure
	If Abnormal Condition selected and there is an uninterpreted event 6	-> add abnormal condition corresponding to the uninterpreted event and associated recovery procedure
Several Matches	*General* 7	-> change recognized expected condition criticality -> modify recognized expected condition
	If two identical expected conditions at two different hierarchical level 8	-> remove one expected condition if expected twice -> modify one of them Advice to remove one goal: - keep the high level goal if the user wants to be able to skip all the lower level procedures when the high level goal is satisfied - keep the low level goal if the user only wants to skip the sub-procedure with the low level goal

Figure 8: Failure types - preferred repairs Table.

[5]This is a first approximation of which repair actions would be needed. We plan to propose more help to the user by suggesting specific procedure components on which those actions should be applied.

Next, PARC assists the user choose the repair among all possible repair options by suggesting a subset of preferred repairs and highlighting the corresponding buttons. Heuristics are used to determine the subset of preferred repairs given the failure type. This heuristic guidance encodes knowledge from the knowledge engineer. The types of failures currently identified and their associated preferred repairs are given in the table below (Figure 8). The type of a failure is determined by PARC using the current state of execution and the pattern matching result. It does not depend on any domain knowledge.

We illustrate two simple cases from the table. In case (3), if none of the expected conditions matches a perceived event (a *surprise*) and if this event is judged significant by the user, PARC suggests that the user adds a new expected condition describing this event and an associated procedure specifying how to recover from it, or to modify an existing expected condition. In case (7), if the recognized expected conditions are correct but the wrong one was suggested by the control assistant, PARC suggests that the user increases the degree of criticality of the right condition (so that it becomes more critical than all the other ones recognized), in order for this condition to be correctly selected the next time and in the same situation. If the user does not modify the criticality order before selecting the right condition, the procedure base is not modified and the same failure may happen again.

In our example, the control assistant has selected the wrong abnormal condition "obstacle" due to the sensor breakdown. It corresponds to case (6) in the table. The heuristic guidance suggests that for this type of failure, an additional abnormal condition describing the unexpected event (force-torque sensor not working) is needed, with its associated recovery procedure. The user selects this repair and the procedure acquisition assistant presents a dialog which leads her through the steps of the repair (cf Section 4).

3.4. Refining the heuristics for preferred repairs

Heuristic guidance, described in the previous section, assists the user in diagnosing a failure and choosing a repair. However, it is based on knowledge elicited from the knowledge engineer and thus may be incomplete. Moreover, the diagnosis stage corresponds to the complex process of explaining failure in the domain in terms of the problem-solving components. Thus, heuristic guidance may also lack some domain specific knowledge, difficult to acquire without actually running the dynamic system. In order to solve those problems, we provide an extra level of automation which improves and refines this heuristic guidance over time with user feedback.

The preferred repairs associated to a given type of failure will be reinforced when actually selected by the user, or extended when the user selects another repair. The reinforcement increases the weight of a link which associates preferred repairs with a failure type. Contextual domain knowledge including for example, the particular task, user profile, type of sensors used and their reliability, and the active procedures, can be also be acquired for each link (as in Boy ,1990).

In our example, we suppose that the user has performed the suggested repair, selected the new expected condition and resumed execution. Later on in system operation, the user finds that once again, when the force-torque sensor was not working, the control assistant selected the "obstacle" abnormal condition. This failure now corresponds to case (7), the control assistant recognized both conditions "obstacle" and "force-torque sensor breakdown," and selected the most critical: "obstacle". In this case, the procedure acquisition assistant suggests modification of the criticality order between those conditions. Thus, the user notes that the heuristic guidance for the previous failure was incomplete, and that the criticality order of the two abnormal conditions needed to be modified as well The user's feedback on the repair choice is used to modify and reinforce the corrected heuristic.

4. AUTOMATING THE REPAIR STAGE

After describing the various levels of assistance available to the user during the diagnosis stage of the procedure refinement task, we now present the automated KA tools useful for the repair stage. Once the user has chosen a repair, she has the choice between directly editing procedures or being guided

further by the procedure acquisition assistant through the planning and performance of the repair. We first describe the lowest level of assistance available, the editors, then the repair guidance.

4.1. Performing the repair

Basic capabilities for editing procedures have been implemented using structured editors. They include the Procedure, Condition, Elementary Condition and Elementary Action Editors (Figure 9). Those editors are directly accessible through the main menu. Editors are *structured* based on the chosen procedure representation (block representation) and integrate a number of syntactic constraints. For example, the user's selection of sub-procedures to be added to a procedure is restricted to the list of existing procedures at a lower hierarchical level than the level of the edited procedure. The Elementary Action editor has been designed to make the model of the domain more transparent to the user. In order to create or modify an elementary action, the user directly "executes" this action on the user interface for control, by selecting a button or setting some numerical parameters for example. The procedure acquisition assistant automatically displays the expression of this action into the Action editor. This is possible since procedures are expressed as user's actions on the interface. For the Elementary Condition editor, all the objects in the domain model are accessible through selection lists to the user.

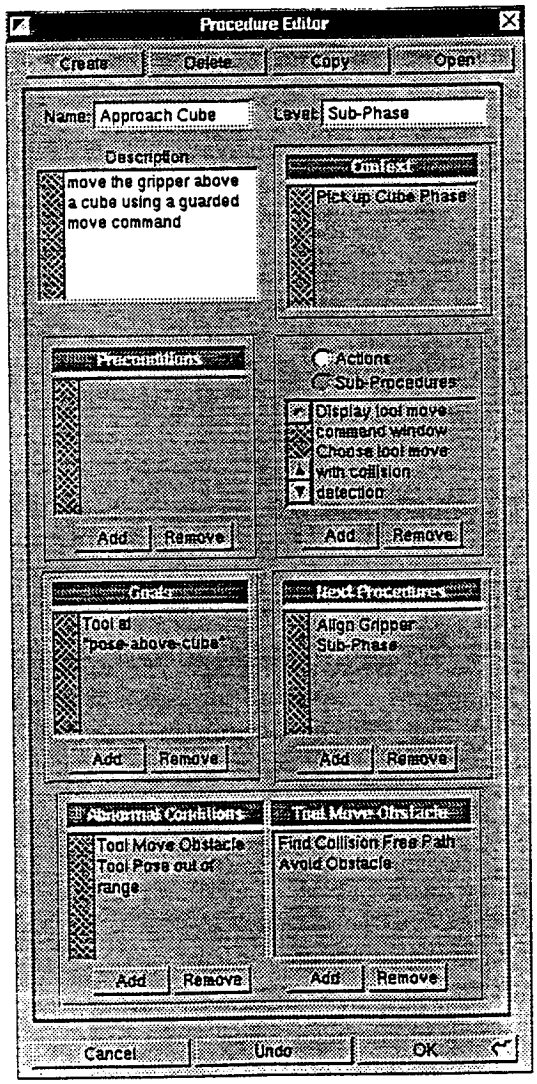


Figure 9: Procedure Editor.

ORIGINAL PAGE IS
OF POOR QUALITY

In our example, the user decides to use the editors to add an abnormal condition describing the sensor breakdown event. She must first access the Procedure editor and display the "Approach Cube" procedure, then add the abnormal condition. The Procedure editor will display the list of existing abnormal conditions she may choose from. If the "sensor breakdown" condition doesn't exist, she must access the Condition Editor, and eventually the Elementary condition editor. Then she goes back to the Procedure editor and add the newly created abnormal condition in the ordered list of abnormal conditions of the "Approach Cube" procedure. She may then associate a recovery procedure with this new condition. If the procedure for approaching a cube without collision detection doesn't exist, she must first create it from scratch, then link it to the "sensor breakdown" condition in the "Approach Cube" procedure. This completes the repair.

Those editing capabilities should be sufficient for the knowledge engineer who knows which editing actions are needed for a given repair and in which order they should be performed. However they do not provide assistance to the domain expert or the operator for planning the repair.

4.2. Planning for the repair

In order to provide more assistance to the user for planning and performing the chosen repair, PARC includes a procedure acquisition dialog which guides the user through the repair stage. It uses knowledge on how to translate a repair into a sequence of editing actions. It also takes advantage of the current execution state to narrow the set of options proposed to the user. Contrary to the heuristic guidance used during the diagnosis stage, this knowledge is mostly procedural and consists of more or less complex repair procedures associated to each possible repair (Mathé, 1991). These repair procedures do not depend on the domain, but only on the knowledge representation syntax and semantics and on the current set of active procedures. Once the user chooses a repair, the appropriate repair procedure is selected and drives the dialog with the user. During this dialog, the user may be asked to describe how external conditions have changed, to specify new procedures, or to state which recovery procedure to associate with an abnormal condition and which active procedures and goals to drop. At some point during this dialog, the procedure acquisition assistant will also choose and display the appropriate editor to the user, where some slots have already been automatically filled in (deduced from the dialog with the user).

Once the repair has been completed or when the dialog is over, the Situation Description panel is updated to reflect the new state of the procedure base. The user may then select the appropriate expected condition and continue the execution. It is important to notice that the user doesn't have to perform a complete repair and may exit the dialog whenever she wants. Since the method is incremental, another related problem will be detected later on during the execution, or again in the same situation. Thus the user can choose how much to modify the procedure base each time she disagrees with the control assistant.

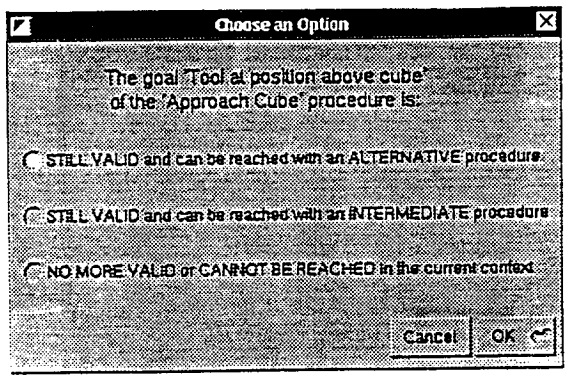


Figure 10: Part of the Repair dialog.

In our example, the user chooses to add an expected condition describing the sensor breakdown event. The following dialog guides her through the repair:

(1) PARC first asks the user to select a condition from the list of existing conditions or to edit a new one. If the "sensor breakdown" condition doesn't exist, PARC displays the Condition editor to the user.

(2) Given the current state of execution, PARC asks the user whether she wants to add this new condition as a goal or an abnormal condition. The user chooses to add an abnormal condition.

(3) PARC asks the user to select from the list of active procedures the procedure to which "sensor breakdown" will be added. The user selects the "Approach Cube" procedure. Then PARC display the list of existing abnormal conditions for this procedure and ask the user to insert the new condition in this ordered list. She inserts it after "obstacle".

(4) The procedure acquisition assistant finally asks the user to select an associated recovery procedure from the list of existing procedures at the same or higher hierarchical level, or to create a new one. The user chooses to create a new recovery procedure.

(5) To define this recovery procedure, PARC first asks the user to specify if the goal "gripper at position above cube" of the "Approach Cube" procedure may still be reached by a local recovery procedure when the force-torque sensor is down. If the goal is still valid, PARC also asks the user if the recovery procedure is an alternative procedure or an intermediate procedure (to be executed before going back to the current procedure "Approach Cube"). The user answers that an alternative procedure is needed (Figure 10).

(6) From this information, PARC automatically deduces that the new alternative procedure's goal is "gripper at position above cube," pointing towards the same next procedure "Align Gripper," and belonging to the same context "Pick up Cube". It displays this information in the Procedure editor, and the user only needs to enter the new procedure name and actions needed for approaching a cube with no force-torque sensor.

When the repair is done, the procedure base is updated (Figure 11), and the Situation Description panel graphically displays the new expectation "sensor breakdown" attached to the "Approach Cube" procedure and its associated new recovery procedure. The user may then choose this condition and continue the execution.

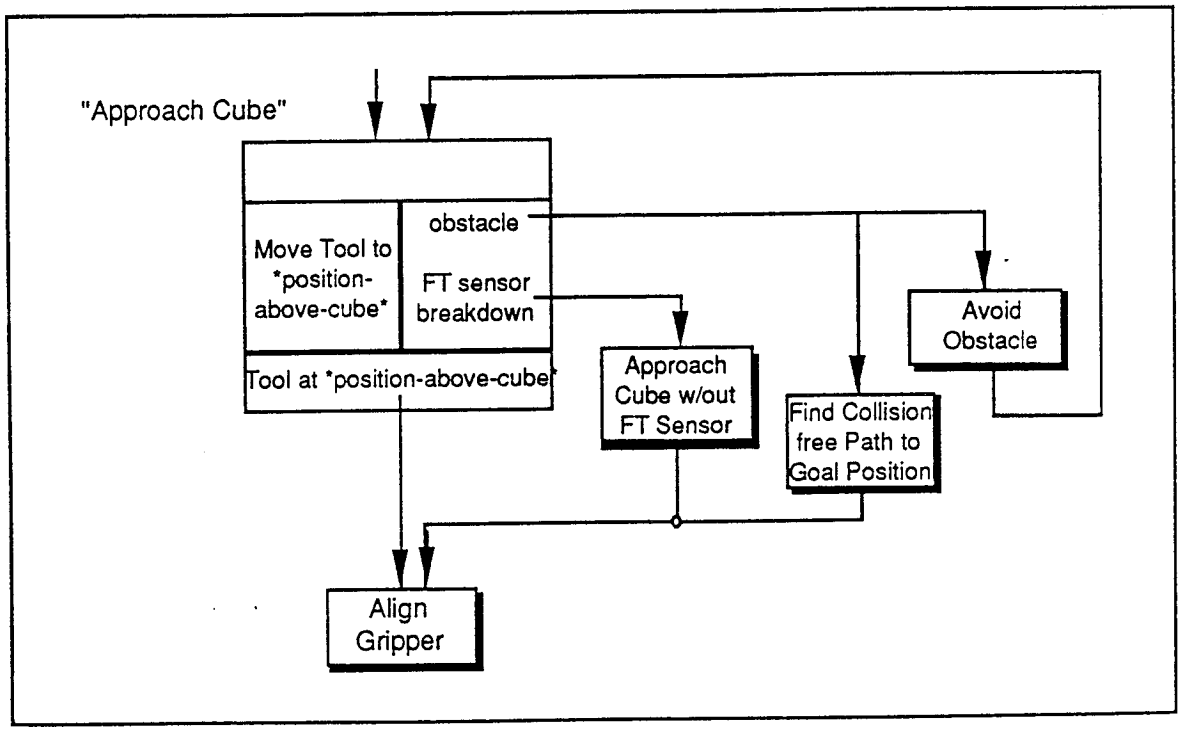


Figure 11: Part of "Pick up Cube" procedure after repair.

5. DISCUSSION

In Sections 3 and 4 we presented the procedure acquisition assistant guiding the user in the diagnosis and repair of procedures. We demonstrated that PARC is able to assist the user in all four steps of the procedure refinement task by providing increased levels of assistance. The modules of the control assistant and of PARC are at various stages of completion. The control assistant, which includes a friendly user interface, is currently operational. It is able to control the Puma 560 robot arm for the robot assembly task. The structured editors and display of current execution state in PARC are implemented. The procedure acquisition dialog, and the heuristic guidance for diagnosis and repair, are designed and currently being implemented. We are currently in the process of completing the design for the reinforcement of heuristics from user feedback. Modules are implemented in C, Objective C, Common Lisp on Sun and NeXT workstations, and communicate through the Task Control Architecture (Fedor & Simmons, 1991).

We now analyze PARC in terms of the taxonomy for interactive knowledge acquisition tools proposed by Musen (1989). Instead of placing PARC at a fixed level in the taxonomy, PARC can be viewed as moving through various levels of the taxonomy with increased assistance. At the lowest level of assistance, the structured editors in PARC require knowledge of the domain, the problem-solving method, and the knowledge representation. Thus, at this level PARC can be seen as a "method level" tool, useful for knowledge engineers. However, the heuristic guidance assists the user in diagnosing and repairing a failure - automating the process of expressing domain knowledge in terms of components of the problem-solving method. Thus, this increased assistance makes the knowledge level analysis more transparent to the user. Finally, PARC already includes domain specific knowledge on the task and the robotics domain. Moreover, the refinement of the heuristic guidance with user feedback can be seen as tailoring the guidance to the user and the task by acquiring domain dependent knowledge, making PARC a "task level" tool, potentially more usable to domain experts.

Another important aspect of PARC concerns its representation for procedures, which can be seen an an intermediate representation for communicating with users in reactive control domains. The block representation narrows the gap in representation mismatch. Representation mismatch (Buchanan *et al.*, 1983) refers to the gap between the user's knowledge, and the knowledge needed in order to communicate with the KA system. In PARC, most steps in procedure acquisition require that the user be familiar with the reactive control method and the procedure representation. Does that present the typical user with too great of a representation gap? This can only be answered definitively through experimentation with PARC. However, we have reason to believe that reactive control method and the block representation do not present much of a gap because they were developed as natural representations for operators of dynamic control systems. In particular:

(1) We designed this model following a cognitive analysis of the robotic control task;

(2) This model is confirmed by studies in other domains involving a reactive control task;

(3) In reactive control domains, operational knowledge is represented in manuals using terms such as "procedures," "nominal," "emergency situations," "recovery procedures," etc., whose semantics are well known to the domain experts (both system designers and operators);

(4) From our experience on two different robotics tasks, the elements of the reactive control method seem to be familiar and intuitive to the domain experts.

6. RELATED WORK

Knowledge acquisition can be broadly classified into three stages: *elicitation*, *refinement* and *restructuring* (Bareiss *et al.*, 1989). Currently we have automated only the *refinement* stage of the procedure acquisition process. The *elicitation* and *reformulation* stages are done manually. Related work on procedure elicitation automation has focused on providing graphical tools to the domain expert to assist her in encoding complex procedures with a visual language (Puerta *et al*, 1991; Saito *et al.*, 1991). Those graphical tools are very useful, but they need to be extended to facilitate the visualization of complex procedures during execution and repair. In particular, those approaches are based on flow diagrams. We believe that a more specific visual language is needed to represent procedures for reactive

control. Most work on the refinement stage has been done for rule-based expert systems performing heuristic classification (Davis, 1977; Araki *et al.*, 1991). Some more recent work tries to automate the refinement process by including explanation-based learning technique (Pazzani *et al.*, 1991). Gruber (1989) proposed a method for refining strategic knowledge during reactive planning. This method is based on the induction of rules from examples, and is different from our method based on the block representation.

Reformulation has received little attention. In PARC, reformulation would enable the user to modify the structure of the procedure base into contexts of use. Such a context structure elicitation is actually done manually by interviewing the system designer. Grant (1991) has shown that the context structure for complex control tasks is incrementally built during system operation. In PARC, context restructuring is currently left to the user through the structured editors. However, Boy (1989) has demonstrated that the block representation could support the automation of such a restructuring process (Boy, 1989).

There have been recent attempts to integrate machine learning and knowledge acquisition methods (Wilkins, 1991). In our investigation, we integrated some ideas from machine learning for automated refinement of plans in reactive systems (Kedar *et al.*,1991) with our method. Most other machine learning efforts has investigated the problem of knowledge base and procedure refinement independently, under a number of guises. The approaches to refinement of knowledge bases in expert systems (e.g. Politakis & Weiss, 1984) typically perform induction over cases. Purely inductive approaches to refining procedures for dynamic systems include reinforcement learning (Lin, 1990) and active experimentation (Gil, 1991; Christiansen *et al.*, 1990). Machine learning approaches that are not purely inductive (i.e. using a preexisting knowledge base) fall into four broad categories: (i) explanation-based learning from failure, which uses a complete/correct knowledge base is to refine the incomplete/incorrect knowledge base (e.g. Hammond, 1986; Chien, 1989); (ii) induction and active experimentation when the explanation is insufficient (e.g. Rajamoney & DeJong, 1988; Pazzani *et al.*, 1991); (iii) acquisition through apprenticeship learning from the user (e.g. Laird *et al.*, 1990; Wilkins, 1988) and (iv) using one approximate knowledge base to refine another (e.g. Kedar *et al.*, 1991).

7. FUTURE WORK AND CONCLUSION

In order to evaluate PARC, a set of experiments will be conducted this year with users in the robotics domain. The following hypotheses are among those we plan to test: (1) Does PARC increase the accuracy and/or efficiency of procedure refinement (over manual refinement, in comparison with other procedure refinement tools)? (2) Does each level of assistance provide increased accuracy and/or efficiency? (3) Is the procedure representation appropriate for the typical users of this system? In order to perform these experiments, a rigorous measure of 'accuracy' and 'efficiency' of the procedure base needs to be defined and measured (using the number of execution failures, aborted goals, execution time, repeated failures, etc.). We also need to define metrics of comparison among different refinement methods.

While our procedure acquisition method is not dependent on the specifics of the robotics domain, it currently addresses only some of the issues in supporting procedure management and maintenance for complex systems. In particular, this methodology assumes that procedure refinement is done as an individual task, and that time is available to interact with PARC to correct the procedure base. This makes it more suited to development of procedures on simulator by small teams. More work needs to be done to study how procedures are designed and refined by corporations such as airlines companies, where operators are only remotely involved in the process (Degani *et al.*, 1991). Furthermore, tools for evaluating the consequences of choosing one procedure over another or of modifying a procedure could be very helpful during the refinement process.

We are at the preliminary stages of implementing the heuristic guidance. The solution proposed only acquires new links or reinforces/weakens existing links between failures types and repairs, but it cannot currently acquire new failures types. Only some of the failures types could be acquired from the user. In particular, those are failure types that can be discerned directly from the graphical representation of the current execution state (e.g. a missing link between an abnormal condition and a recovery

procedure for it). We are currently considering a technique similar to the one proposed in (Gruber, 1989) for acquiring rules associating new failures with repairs by induction from examples of failure types and repairs.

We plan to improve the flexibility of PARC. Currently, the levels of assistance are hard-coded in PARC. PARC could be made more modular, letting the user select the levels of automation during the procedure refinement process. A user-profile could also be used to choose a default level of assistance for different types of users.

We also plan to enhance the user interface for control by adding several features: higher level sensor predicates more useful to the user, and a 3D simulator (with special glasses) in order to simulate a new procedure before executing it with the real robot.

Finally, we plan to develop a graphical visual language for the procedure representation, assisting the user in exploring the procedures base, displaying an execution trace in the procedures network, and letting the user edit procedures graphically.

We hope that others will join us in the effort.

Acknowledgements

Many thanks to Guy Boy and Lonnie Chrisman for their insightful comments on this project. Thanks to Jay Steele for developing the robotics environment and to Jonathan Kolyer for developing the user interface on the NeXT. Thanks to Reid Simmons and Chris Fedor of Carnegie-Mellon University for providing and supporting the Task Control Architecture. Thanks to Guy Boy, Peter Friedland, Kate McKusick (NASA Ames) and Brian Slator (ILS) for reviewing this paper.

References

Araki, D., Kojima, S. (1991); *KASE Project towards Effective Diagnosis System Developments* ; Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 1, Banff, Canada, October 6-11.

Bareiss, R., Porter, B.W., Murray, K.S. (1989); *Supporting Start-to-Finish Development of Knowledge Bases* ; Machine Learning, 4(3-4):259-283.

Boy, G.A. (1989); *The Block Representation in Knowledge Acquisition for Computer Integrated Documentation* ; Proceedings of the 4th AAAI-Sponsored Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, Oct. 1-6.

Boy, G.A. (1990); *Acquiring and Refining Indices According to Context* ; Proceedings of the 5th AAAI-Sponsored Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, Nov. 4-9.

Buchanan, B. G., Barstow, D. K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. A., (1983); *Constructing an Expert System.* In F. Hayes-Rogh, D. A. Waterman, D. B. Lenat (eds.), *Building Expert Systems.* Reading, MA: Addison-Wesley.

Chien, S. A. (1989); *Using and Refining Simplifications: Explanation-Based Learning of Plans in Intractable Domains.* Proceedings of the Eleventh IJCAI, Detroit, MI.

Christiansen, A. D., Mason, M. T., and Mitchell, T. M. (1990). *Learning Reliable Manipulation Strategies without Initial Physical Models.* Proceedings of the IEEE International Conference on Robotics and Automation.

Davis, R. (1979); *Interactive transfer of expertise: Acquisition of new inference rules* ; Artificial Intelligence, 12(2):121-157.

Degani, A., Wiener, E. L. (1991); *Philosophy, Policies, and Procedures: The Three P's of Flight-Deck Operations* ; Proceedings of the 6th Int. Symposium on Aviation Psychology, Columbus, Ohio, April 29- May 2.

Fedor, C., & Simmons, R. (1991); *Task Control Architecture User Manual* ; Manual Version 5.2, Carnegie-Mellon University, June 1991.

Gil, Y. (1991); *A Domain-Independent Framework for Effective Experimentation in Planning;* Proceedings of the Eight Workshop on Machine Learning, Evanston, IL, June 27-29.

Grant, S.A. (1991); *Modelling Cognitive Aspects of Complex Control Tasks* ; Ph.D. Dissertation, 1991, University of Strathclyde, UK.

Gruber, T.R. (1989); *Automated Knowledge Acquisition for Strategic Knowledge* ; Machine Learning, 4(3-4):293-336.

Hammond, K. J. (1986); *Learning to Anticipate and Avoid Planning Problems Through the Explanation of Failures ;* Proceedings of the Fifth AAAI Conference, Philadelphia, PA.

Kedar, S., Bresina, J., Dent, L. (1991); *The Blind Leading the Blind: Mutual Refinement of Approximate Theories* ; Proceedings of the Eigth Workshop on Machine Learning, Evanston, IL, June 27-29.

Laird, J. E., Hucka, M., Yager, E. S., and Tuck, C. M. (1990); *Correcting and Extending Domain Knowledge Using Outside Guidance.* Proceeding of the Seventh International Machine Learning Conference, Austin, TX.

Lin, L. (1990); *Self-Improving Reactive Agents: Case Studies in Reinforcement Learning Frameworks;* Proceedings of the International Conference on Simulation of Adaptive Behavior, Paris, France.

Mathé, N. & Boy, G.A. (1992); *The Block Representation for Procedure Acquisition* ; Proceedings of the AAAI-92 Workshop on Knowledge Representation Aspects of Knowledge Acquisition, San Jose, CA, July 16.

Mathé, N. (1991); Second Progress Report on "Procedure Management and Maintenance"; European Space Agency internal report, August 31.

Mathé, N. (1990a) ; *A Blackboard Approach to Intelligent Assistance for Space Telemanipulation* ; AAAI-90 Workshop on Blackboard Systems, Boston, Massachusetts, July 29.

Mathé, N. (1990b) ; *A Space Remote Control Application : Cognitive Modeling and Blackbaord-Based Implementation* ; Proceedings of the 3rd International Conference on Human-Machine Interaction and Artificial Intelligence in Aeronautics and Space, Toulouse, France, Sept. 26-28, pp 377-392.

Mathé, N. (1990c); *Intelligent Assistance for Process Control: Application to Space Teleoperation* ; Thesis Dissertation (in french), Nov. 90, ENSAE, Toulouse, France.

Musen, M. A. (1989); *Conceptual Models of Interactive Knowledge Acquisition Tools* ; Knowledge Acquisition, 1(1):73-88.

Musen, M. A., Fagan, L.M., Combs, D.M., & Shortliffe, E.H. (1987); *Use of a domain model to drive an interactive knowledge editing tool* ; International Journal of Man-Machime Studies, 26(1):105-121.

Nielsen, M., Grue, K., Lecouat, F. (1991); *Expert Operator's Associate: a Knowledge-Based System for Spacecraft Control* ; Proceedings of the 1991 Goddard Conference on Space Applications of AI, Greenbelt, Maryland, May 1991.

Pazzani, M.J., Brunk, C.A. (1991); *Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning* ; Knowledge Acquisition, 3:157-173.

Politakis, P. and Weiss, S. M. (1984); *Using Empirical Analysis to Refine Expert System Knowledge Bases;* Artificial Intelligence 22(1):23-48.

Puerta, A., Egar, J., Tu, S., Musen, M. (1991); *A Multiple-Method Knowledge-Acquisition Shell for Automatic Generation of Knowledge-Acquisition Tools* ; Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.

Rajamoney, S. A. and DeJong, G. F. (1988); *Active Explanation Reduction: An Approach to the Multiple Explanations Problem;* Proceedings of the Fifth Machine Learning Conference, Ann Arbor, MI.

Saito, T., Ortiz, C., Loftin, R.B. (1991); *On the Acquisition and Representation of Procedural Knowledge* ; Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.

Wilkins, D. C. (1988); *Knowledge Base Refinement Using Apprenticeship Learning Techniques;* Proceedings of the Ninth AAAI Conference, St. Paul, MN.

Wilkins, D.C. (1991); *A Framework for Integration of Machine Learning and Knowledge Acquisition Techniques* ; Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Vol 2, Banff, Canada, October 6-11.