Impasse-Driven Tutoring for Reactive Skill Acquisition

Randall W. Hill, Jr.
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 525-3631
Pasadena, CA 91109, USA
hill@gobi.jpl.nasa.gov

W. Lewis Johnson
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292, USA
johnson@isi.edu

1. Introduction

We are interested in developing effective performance-oriented training for the operation of systems that are used for monitor and control purposes. We have focused on one such system, the communications Link Monitor and Control (LMC) system used in NASA's Deep Space Network (DSN), which is a world wide system, for navigating, tracking and communicating with unmanned interplanetary spacecraft. The tasks in this domain are procedural in nature and require reactive, goal-oriented skills; we have previously described a cognitive model for problem solving that accounts for both novice and expert levels of behavior as well as how skill is acquired [Hill and Johnson, 1993]. Our cognitive modeling work in this task domain led us to make a number of predictions about tutoring that have influenced the design of the system described in this paper,

Tutoring in our training system is *impasse-driven*. Unlike other tutoring techniques such as *model tracing* and *plan recognition*, which decide to intervene on the basis of understanding every student action, our approach depends instead on impasse recognition to drive the tutoring intervention. When the tutor recognizes that the student has reached a procedural impasse, it intervenes with advice that it generates with its expert cognitive model, which is used to understand and resolve the current impasse in situ.

The tutor uses a method called *situated plan attribution* to recognize and explicate students' procedural impasses in the LMC task domain. We use the term *plan attribution* instead of *plan recognition* because it does not assume that the problem solver's actions are controlled by plans. Rather, we take the situated action view of plans: plans are resources that guide and orient action, and actions are ultimately contingent on the state of the world [Suchman, 1987].

The tutor recognizes potential impasse situations by using a number of different resources. Plans are attributed to the student based on a description of the task. The attributed plans, in turn, generate expectations about the student's actions and goals. Each action is evaluated with respect to the student's attributed plans, its actual effects on the training device and the goals associated with these plans. Impasses related to specific actions are often indicated by feedback from the device simulator, e.g., a command is rejected or a warning is given. This type of impasse is easy to detect, both for the student and the tutor, though it is not always easy to resolve. Other impasses, are not as obvious, since the student's action may result from a misconception about a plan or goal rather than violating a constraint of the system, and the error may not manifest itself for some time. For instance, the student may complete a procedure without achieving it's goals or take an action that is not relevant to the current task. The tutor recognizes this type of error as a *potential* impasse since the device itself may not show any sign of malfunctioning and the student may be unaware of a problem until a much later stage in the task. When the tutor detects a potential impasse it intervenes, thereby forcing the impasse in order to resolve the misconception near the point at which it showed itself.

We have implemented our tutor in Soar [Laird et al., 1987], an integrated problem solving and learning architecture. Our tutor uses the Soar chunking mechanism to learn and is thus often able to recognize an

impasse and its resolution in the task context without searching through a set of problem spaces to re-derive a solution.

2. Task Domain: Monitor and Control Systems

Monitor and control systems are ubiquitous in our increasingly automated society: power plants, factories, environmental control systems and operations centers all use computers to control other machines and to monitor their status and health. Typically only a portion of the work to be done in the problem domain is automated by the monitor and control system, leaving tasks requiring judgment and specialized knowledge to a human Operator. [1] In this paper we describe an approach to training Operators that has been developed for tasks in one such domain, namely, the communications Link Monitor and Control (LMC) system used in NASA's Deep Space Network.

The DSN is a worldwide system for navigating, tracking and communicating with all of NASA's unmanned interplanetary spacecraft. As the name suggests, the LMC system is used by operations personnel to monitor and control a DSN communications link during an interaction with a spacecraft. A communications link is formed by assigning a collection of equipment to an LMC system for a particular mission; the link typically includes a large antenna dish (34 or 70 meters in diameter), its electromechanical controllers and subsystems, a receiver/exciter, a digital spectral processor (DSP), a precision power monitor (PPM), a hydrogen maser, frequency and timing subsystem (FTS), phase calibration generators, digital tone extractor (DTE), and numerous other devices.

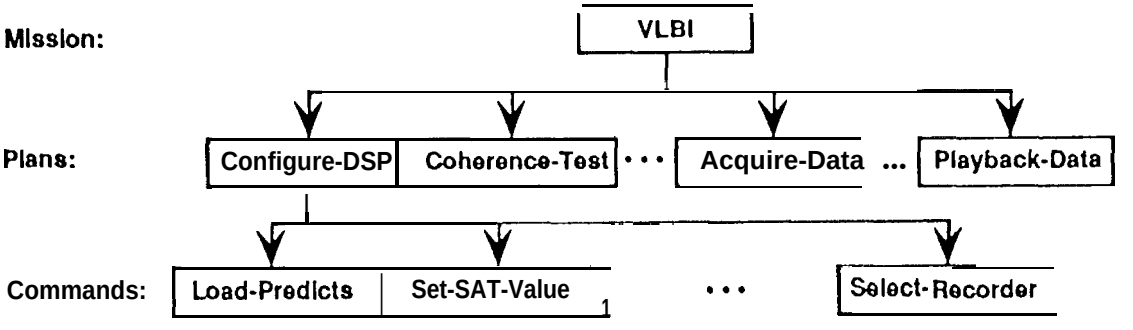

Figure 1 The VLBI task with a representative set of plans and operators

The tasks performed by LMC Operators involve configuring and calibrating the communications link, testing the configuration for coherency, acquiring data from the spacecraft and playing back the data to the network operations control center. Each mission consists of a number of other similar tasks, and the way to perform each task is described by a procedure[2] manual, which provides the Operator with command sequences for each of the various subsystems. Figure 1 shows a portion of the task hierarchy for a VLBI[3] mission. Note that the figure shows three basic levels in the hierarchy: mission, *plan*, and *command*. What Figure 1 does not convey is that the order in which the plans and commands are executed is not completely specified; though some partial orderings exist among plans and commands, there is a lot of room for variability from one mission to the next (or from the way one Operator performs the tasks to the next.)

To accomplish a VLBI mission, the Operator performs the plans that are shown in Figure 1: Configure-DSP, Coherence-Test, and so on. The Configure-DSP plan has operators[4] to load the mission-specific pre-

1 The term *Operator* will be used two different ways in this paper. To disambiguate its Usage, we capitalize *Operator* when referring to a human being, and we use lower case to refer to a Soar *operator*.

2 Henceforth we refer to these written procedures as plans.

3 VLBI stands for Very Long Baseline Interferometry.

4 An operator with a small . . . denotes a function that is selected and applied to a state to get a result. This is in keeping with the Soar notion of an operator [Laird *et* al., 1987].

diction data file (Load-Predicts), set the attenuation values on the Intermediate Frequency Video Down Converter (set-SAT-Values), ... and select a recording device to capture the mission data (Select-Recorder). Applying an operator involves issuing a command (e.g. the Load-Predicts directive is NLOAD *predicts-file).*

It would appear that the tasks in the LMC domain are straightforward and would require little or no training - just follow the plans given in the mission procedure manuals. We have found, however, that this is not the approach taken by domain experts. Through extensive interviews with expert operators and system engineers, we determined that the procedure manuals only provide a subset of the knowledge needed to successfully perform the tasks associated with a mission. What is generally lacking in the procedure manuals is a complete description of the required device state conditions before and after a command is issued. Expert Operators possess a knowledge of the preconditions and postconditions for each command and verify these conditions are satisfied before and after issuing the commands. Operators who lack this knowledge may find it difficult to complete even simple plans, since the commands may be rejected, or worse, they may put the device into an incorrect state for the current plan or mission. For example, one of the preconditions for the Load-Predicts operator is that the predicts-file being loaded must be present on the system. If the Load-Predicts command is issued for the predicts-file named "JK" (i.e., the Operator issues NLOAD JK), it will be rejected if a file by that name is not present in the predicts file directory.

To complicate matters, during a two hour mission an Operator may interact with five major subsystems, comprised of fifty different devices, for which more than **250** unique attributes must be monitored and 500 event notice messages processed. The sheer quantity of monitor data accentuates the difficulty of executing the control procedures. Unexpected device state changes, device failures, and the slow reaction time of certain devices can cause procedural impasse and Operator confusion. If a command is issued when one of its preconditions is not satisfied, then it is likely to be rejected, or worse, it puts the device into an undesirable state. When Operators observe that a precondition is not satisfied, they have to know how to react. Consequently, procedurally-defined command sequences are not sufficient to accomplish most task goals. The plan acts as a guideline, but the Operator must bring other knowledge to bear on the performance of a task.

3. Domain Problem Solving: Implications for Plan Recognition

In this section wc begin to motivate our approach to tutoring by first describing the nature of problem solving and skill acquisition in the LMC domain. In order to better understand the nature of these skills we implemented a detailed cognitive model that accounts for the behavior of both novice and expert LMC Operators. Developed in Soar, a problem solving architecture with a learning capability [Laird et al., 1987], the model enabled us to make a number of predictions about how Operators acquire knowledge and skill in this domain; it improves its behavior over time, acquires new knowledge and is able to recover from incorrect knowledge [Hill and Johnson, 19931. We describe how these predictions about skill acquisition affected the design of our intelligent tutor. Ultimately, we desired to capitalize on the insights gained from the cognitive model, so we start by revisiting some of the key issues raised by the cognitive model that motivated our approach to tutoring using *situated plan attribution.*

3.1 Plan Execution and Situated Action

Problem solving in the LMC domain involves both plan execution and situated action. By plan execution we mean that Operators issue commands according to a written procedure. Actions are situated, on the other hand, in that the context in which commands are issued must be taken into account, resulting in behavior that does not always resemble the original plan. A plan may specify a command to be executed, but the situation may warrant taking a different action in order to achieve the underlying goals of the plan or task. To illustrate the effect of the device situation on the Operator's actions, consider the case where there are two plans that need to be executed as a part of the VLBI task: first the Configure-DSP plan and then the Coherence-Test plan. The tables in Figure 2 show the default command sequences for these two plans, where the operators are shown in the left column and the corre-

sponding commands are shown in the right column. The Utility Commands table contains commands that are not typically used in any particular plan, but arc useful for changing a device's state.

Configure-DSP	
Operator	Command
Load-Predicts	V NLOAD <set-id>
Set-S-Attenuation	V SAT <sat>
Set-X-Attenuation	V XAT <xat>
System-Temperature	V NTOP <s><x>
Select-Recorder	V NRMED <nrmod>
set-offset	V OFST <ofst>

Legend:
V . Subsystem ID, <> = parameter variable

Coherence-Test	
Operator	Command
Phase-Calibration-Gen	V NPCG <v1>
Run-NCB-Program	V NRUN <v2>
Run-FFT-Program	V NFFT <v3>
Digital-Tone-Extractor	V NDTE <v4>

Utility Commands	
Operator	Command
Idle-NCB-Program	V NIDLE REC

Figure 2 Plan Descriptions for Configure-DSP and Coherence-Test . A utility command is also shown that is not associated with a particular plan.

```
> V NLOAD JK
> COMPLETED. LOADING NCB PREDICTS . . .
>VSAT25
> COMPLETED. S-BAND ATTENUTATION=25 DB
>VXAT20
> COMPLETED. X-BAND ATTENUATION=20DB
> V NTOP 17.319.4
> COMPLETED. SYSTEM TEMP: S 17.3 X 19.4
> V NRMED LD0
> COMPLETED. NRMED: LOO
>VNPCG MAN
> COMPLETED. NPCG MODE: MANUAL
> V NRUN COLD
> COMPLETED. NCB MODE: NRUN
> V NFFT E
> ...
```

Case 1: Two errors: **(1)** wrong SAT value (should be 30 instead of 25) and (2) missing the OFST command.

```
> V NLOAD MK
> REJECTED. NOT ALLOWED IN NRUN MODE
> V NIDLE REC
> COMPLETED. NIDLE REC INITIATED
> V NLOAD MK
> REJECTED. INVALID SET NAME
> ...
```

Case 2: Multiple constraint errors.

```
> COMPLETED. NIDLE REC INITIATED
> V NLOAD JK
> COMPLETED. LOADING NCB PREDICTS
> V SAT 30
> COMPLETED. S-BAND ATTENUTATION=30 DB
```

Case 3: Avoiding the Case 2 errors.

Figure 3 Operator Logs: Examples of Errors and Situated Action

Figure 3 provides three different traces of an Operator executing these plans. The traces show the commands in the sequence they were issued by the Operator, where each command is followed by a message from the device telling whether the command was accepted or rejected.[5] Case 1 contains two typical novice errors that could easily go undetected. The first error was that the Operator mis-parameterized the SAT command with a value of 25 instead of 30. Normally the Operator would obtain the SAT value from a calibration table that contains this information for each antenna system, so the error may have been due to mis-reading the table. The second error was the omission of the OFST command, which is the last step of the Configure-DS]' plan; this mistake will not manifest itself to the Operator, rather, it will affect the data correlation later, after the mission is complete.

Case 2 illustrates two examples of how the device simulator responds to constraint violations associated with the NLOAD command. We see that after NLOAD was rejected the first time the Operator recognized the nature of the problem and corrected it with the NIDLE REC command. But when

[5] The COMPLETED response means that the command was accepted by the device, but it does not imply that the intended action was successful. The REJECTED response means that the device will not attempt to execute the command because it violates a system constraint.

NLOAD was re-issued, it was again rejected due to the fact that the predict set specified by the NLOAD parameter, MK, did not exist.

Case 3 shows how a skilled Operator responds to the same situation as Case 2. Instead of rotely following the Configure-DSP plan, the Operator recognized that the situation called for issuing the NIDLE REC command first, and then the NLOAD JK command.

3.2 Cognitive Model

These three cases show the typical situations faced by LMC Operators, where it is not possible or feasible to just execute the plans described by the procedure manuals. Our Soar-based cognitive model was developed to learn how to handle situations of this type, where it executes the known plans until it recognizes that the situation requires a different action than the prescribed one, In cases where its actions failed, it learned from the failure and acquired new knowledge to help it avoid the same error in the future, The details of how the cognitive model performed are described in [Hill and Johnson, 1993], and in the following paragraphs we summarize the conclusions from that effort that had an influence on the design of the tutor described in this paper.

(1) *Learn by doing.* Though this is not a new insight in the field of education, it is one that is strongly supported by the cognitive model and is a direct result of the way that the Soar chunking mechanism works [Newell, 1990]. This result agrees with Anderson's account of proceduralization and skill acquisition [Anderson, 1983]. It is not sufficient to acquire declarative knowledge about a task, rather, knowledge must be brought directly to bear on solving problems. Our cognitive model's task performance improves only as it compiles its knowledge about how to perform the domain task. As a consequence of this result, we have designed our training system to maximize the Operator's experience in performing the target task skills; the Operator performs tasks on a Link Monitor and Control system simulator and the tutor strategically intervenes in a manner that will be described later.

NCB-Program Device	
Attribute	Value
MODE	RUN

VLBI Predicts Set Device	
Attribute	Value
NAME	JK
RECEIVED?	YES
QUALITY?	OK

Figure 4 Partial Set of Devices and their Attributes

Operator	Command	Precondition Name	Device	Attribute	Value
Load-Predicts	VNLOAD <id>	Load-Predicts-PC1	NCB-PROGRAM	MODE	IDLE
		Load-Predicts-PC2	VLBI-PREDICTS	RECEIVED?	YES
		Load-Predicts-PC3	VLBI-PREDICTS	QUALITY	OK

Figure 5 Preconditions for Load-Predicts Operator

(2) *Expert problem solvers do not rotely execute plans.* Rather, they evaluate the appropriateness of executing each command in the plan with respect to the device situation, and they issue a command only when all of its preconditions arc met. Moreover, expert problem solvers recognize when it is necessary to reactively plan in order to overcome unsatisfied command preconditions. To illustrate these points, consider the device model in Figure 4, which reflects the state of the NCB-program and VLBI Predicts Set devices at the time that the Operator performed the actions shown in Figure 3. 'I'he state of a device is represented with a set of attribute-value pairs. In this case, the NCB-Program device's MODE attribute has a RUN value, and the VLBI Predicts Set is named JK, has been received and its quality is acceptable.

The Load-Predicts operator in Figure 5 has three preconditions, of which only two are satisfied with respect to the current device situation. The Load-Predicts-PC1 precondition is not satisfied since it re-

quires the NCB-Program to be in the IDLE mode rather than the RUN mode. According to our cognitive model, an expert Operator will notice that this precondition is unsatisfied and will act to correct the situation, as we observe in Case 3, by issuing the NIDLE REC command to put the NCB-Program into the IDLE mode.

(3) *Novices acquire skill by learning operator preconditions.* In addition to describing expert behavior, our cognitive model also accounts for novice behavior and the process by which skill is acquired. It predicts that a novice Operator who does not know about a precondition that is unsatisfied will make the errors shown in Case 2, issuing the command in the wrong situation, resulting in a rejection. In our cognitive model, the novice acquires skill by recognizing there was an error (i.e., the rejection notice), understanding the meaning of the rejection message, searching for and applying an operator to repair the situation, and adding the newly acquired precondition to its knowledge; these steps correspond to the goals that a student would have in trying to recover from an error or an impasse, and they are also opportunities for a tutor to give assistance. In addition to making errors due to missing preconditions, our model also addresses the situation where the Operator has a misconception about a precondition and needs to learn the correct knowledge.

(4) *Learning* occurs *in impasse situations.* The **cognitive model** performs the task until it reaches a point where it needs additional knowledge to cope with the situation. It acquires knowledge at the impasse point from an artificial tutor, which provides information about the missing or incorrect operator precondition. These impasse points were a natural place to tutor the novice with the needed situational knowledge since the Soar chunking mechanism could immediately transform the declarative tutoring knowledge into procedural knowledge. In addition to the instruction about preconditions, the tutor also gave the novice corrective steps for resolving the impasse.

(5) *Learning occurs in a goal context.* This observation is related to the previous one and it influences the decision to tutor at the impasse point rather than before or after the training session. The Soar chunking mechanism works by summarizing the results of a subgoal in new operators that can be applied under similar circumstances to obtain the same results without needing to search the subgoal problem space again. All learning in the Soar architecture occurs in a goal context and the contents of what is learned also depends on the goal context. These observations led us to predict that the most effective way of tutoring human students is to intervene in the right goal context, that is, when the student has the goal to resolve an impasse. The challenge for the tutor is to recognize when the student is at an impasse point and then provide the appropriate instruction for resolving the impasse.

(6) *Some knowledge gaps are hidden.* If the student only learns about preconditions by violating them and reaching an impasse, then some preconditions may not be learned, The action sequence order may eliminate the need to verify that a precondition is satisfied since the results of one action satisfy the preconditions of the next; as long as the actions are always executed in the same order the precondition will be satisfied and the precondition will remain hidden in the sense that problem solver will not act to verify that it is satisfied. It is sometimes difficult to create device situations that will force the student into a failure impasse for certain preconditions, nevertheless the tutor needs to detect hidden knowledge gaps, perhaps by forcing the student out of a rote action sequence and into situations where the actions are executed in a different order.

3.3 The Tutoring Intervention Problem

We draw two conclusions about tutoring in the LMC domain from the preceding results. First, the model indicates that impasse situations are strategic opportunities for learning. In [Hill and Johnson, 19931 we describe how our cognitive model acquires new knowledge while searching for operators to resolve the impasse. The key to learning in the model was to provide the pertinent information to resolve the impasse when the student was in a problem space that could make use of it. The implication of these observations is that tutors must be capable of recognizing when the student has committed an error or potential error in order to intervene effectively. 'I"his conclusion is supported by [Galdes, 1990], which

provides evidence that human tutors intervene during performance-oriented training only after they have identified definite or potential errors.

The second conclusion from our modeling work is that once the tutor chooses to intervene, it must situate its explanation with respect to the circumstances that created the impasse. The tutor must be able to apply its expert cognitive model at the impasse point, and it must do it in a way that will generate a situation-specific explanation of the impasse as well as a means of resolving it. We call these two issues the *tutoring intervention problem,* and the approach we describe in the next section attempts to address this problem as it pertains to reactive problem solving domains.

4. Situated Plan Attribution

Our approach to addressing the tutoring intervention problem is based on a method we call *situated plan attribution.* We describe in detail how this method is used by our tutor, which is integrated with an LMC system simulator. This section, which is organized into three parts, makes the following points: it (1) presents some assumptions about plans and actions and how they influenced our decision to use impasse recognition to drive the tutoring intervention; (2) describes how situated plan attribution creates expectations about behavior and provides a context for intervention; and (3) describes how the tutor recognizes impasses.

4.1 Assumptions about Plans and Action

Our approach to tutoring makes a key assumption: plans do not determine behavior [Suchman, 1987]. As we have shown in our cognitive modeling work, plans influence behavior, but they act primarily as a resource to help guide the performance of the task. Consequently, plans are useful for creating expectations about behavior, but they cannot always be used to understand and explain it. For instance, some situations call for reactive planning, resulting in actions that, when observed by a tutor, do not fit into the default plan. The tutor needs to be able to recognize whether the student's actions are appropriate in these situations, but it may be very difficult to do so if the action has to be recognized and explained in terms of a known plan. A strict plan recognition approach to understanding situated behavior would require a library of all plans for all situations.

For the link monitor and control domain, wc have a complete set of default plans for performing a mission, but wc believe it would be impractical to represent all of the variations of the plans that would be needed to account for situational differences that arise from the dynamic nature of the problem domain. For instance, in the situation that was described for Cases 1-3 in Figure 3, a variation of the Configure-DSP plan would have to include the NIDLE REC command to cover the situations where the Load-Predicts-PC1 precondition (Figure 5) is not satisfied. The same would hold true for every precondition of every operator in the plan: the situation could potentially force the Operator to deviate from the standard plan every time a precondition was not satisfied, meaning that there would potentially have to be a plan variant for each such situation.

This viewpoint led us to view plans as a way of providing only partial understanding of the student's behavior; more specifically, plans provide a framework for recognizing when the student has reached a potential impasse, Since impasses may result from a combination of student misconceptions and anonm-1ous device states, wc seek to understand the impasse using an attributed plan as a starting point, but not as the only means of understanding student behavior. Besides plans, the tutor also has access to several other resources for interpreting student behavior: it sees the state of the devices (i.e., situation in which the action is taken), it tracks the state of the goals associated with the attributed plans, and it sees the student's actions and the device's response to them. This approach shifts the computational burden away from giving a plan-based account of a student's behavior and toward using plans as one of several resources for the task of recognizing and explaining an impasse. Plans are a convenient way of organizing knowledge about goals and actions related to the task; they are a declarative description of how to perform the task as opposed to being an executable model, and they arc meant to orient the tutor

rather than giving a complete account of how to behave in the current situation, which is consistent with Suchman's view of plans and situated action.

An additional resource for detecting certain types of impasses is the simulator itself, The simulator rejects directives[6] whenever they violate a system-imposed constraint.[7] Directive rejections are cues used by the tutor in deciding whether to intervene: they eliminate the need to guess whether there is an impasse in these instances. Instead, the tutor only has to note that the directive was rejected and then determine the causes for the impasse and resolve it. This eliminates the need to understand every student action, which is normally a requirement for both plan recognition and model tracing approaches to tutoring.

4.2 Situated Plan Attribution

The primary purpose of situated plan attribution is to generate expectations about Operator behavior. These expectations guide the impasse recognition process as well as the tutorial intervention. We begin by describing how plans are represented and used for generating expectations and recognizing impasses.

For each task there is a set of plans, each of which is based on a written procedure, that will accomplish the goals of the task. Examples of two plans are shown in Figure 2. A partial order among the plans for a mission is represented in a structure called a Temporal Dependency Network (TDN) [Fayyad and Cooper, 1992; Hill and Lee, 1992]. The goal of the TDN is to express an order among the plans that maximizes the concurrency of plan execution. A portion of the VLBI TDN is shown below in Figure 7. Note that the plans, Configure-Precision-I'ower-Monitor and Configure-Receiver-Exciter, can be executed concurrently, while Configure-DSP is executed only after these two have both been completed. If two plans can be executed concurrently, it means that their commands may be interleaved in the command sequence without harmful interaction. Plans that have dependencies are placed in succession to one another using the Before and After relations shown in Figure 7.

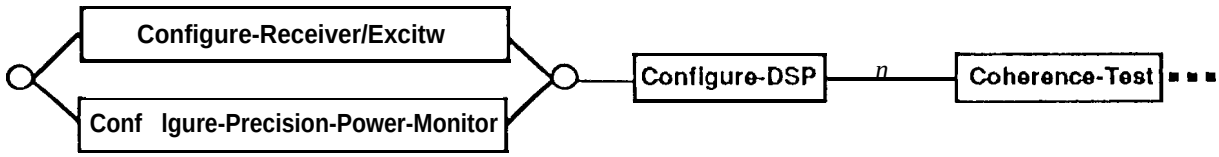

Figure 7 Partial Temporal Dependency Network for VLBI Task

The TDN resembles a procedure net: from' a task perspective it describes a class of plans that would theoretically achieve the task goals. But unlike the procedure net approach to tutoring [Chen et al., 1991; Rickel, 1988; Warriner, 1990], we do not use the TDN as the sole basis for deciding when to intervene and provide tutoring. One difference is the granularity of the descriptions: the TDN specifies the relationship among plans, while the procedure net focuses on the relationships and constraints at the action level. We also loosen the procedure net assumption that actions are plan-based. Whereas the procedure net provides an action grammar for deciding on whether an action is appropriate or not, we use the TDN description to orient the impasse recognition process, which is described in the next section.

Given a mission, the tutor uses the mission's TDN to determine which plans are eligible for execution. These plans, known as *active plans* are the ones that we expect the Operator to be executing. As previously mentioned, multiple plans may be concurrently active, and we expect the Operator to interleave actions from different active plans while performing the various mission tasks. The tutor adds a plan to the active plan set when the plan's predecessors are satisfied and completed. Likewise, when a plan has been evaluated as satisfied and completed, the plan is removed from the active set and placed in

6 Directives are synonymous with commands.
7 The simulator is faithful to the actual link monitor and control system in the way that it accepts or rejects directives.

the inactive set. Once a plan is placed in the active set, it creates expectations not only about what actions will be taken but also which goals will be active. The tutor marks successfully completed actions on the plan, and when all of the expected actions have been observed, the plan is marked "completed". A completed plan is not removed from the active set, however, unless its goals are also satisfied.

The plan's goals, like its actions, are continually monitored by the tutor, beginning at the time that the plan is placed in the active set, 'I'he tutor observes all device state changes and it recognizes when each of the plan's disjunctive goals are achieved. Once the conjunction of all the plan's goals are satisfied, the plan is considered to be satisfied. Figure 8 shows some of the Configure-DSP plan's goals.

Configure-DSP		
Device	Attribute	Expected Value
VLBI-PREDICTS	LOADED?	YES
CONFIGURATION-TABLE	SYNTHESIZER-FREQ-R1	300.21
CONFIGURATION-TABLE	SAT-VALUE	30
CONFIGURATION-TABLE	RECORDER	LD0
CONFIGURATION-TABLE	OFST-VALUE	2.7
...	...	...

Figure 8 Plan Goals for Configure-DSP

4.3 Impasse Recognition

The notion of an impasse is not new: it has been used in *repair theory* to help explain procedural errors in subtraction [Brown and VanLehn, 1980] and as a motivation for subgoaling during problem solving *(e.g.,* **the Soar** architecture [Laird et al., 1987].) Our definition of an impasse is consistent with these uses of the term: impasses are obstacles to successfully performing a procedure, where the obstacle is a lack of knowledge or misconception about what to do next in a task situation.

The impasses recognized by our tutor fall into three categories: *action-constraint violations, goal failure,* and *plan dependency violations.* Impasses in the first category, *action-constraint violations,* arc usually easy to recognize, both by the student and the tutor, because the device simulator gives an indication when a system constraint has been violated. Impasses in the other two categories are different from the *action-constraint violations* in that they arc *potential* rather than *actual* impasses, They do not become actual impasses until the student notices that there is a problem, which may not be for a significant amount of time in these instances. Since we wish the tutor to intervene as near to the commission of an error as possible, the tutor forces an impasse when it detects a goal *failure* or *plan dependency violation*.

One of the significant aspects of our approach is that it does not attempt to recognize the mental state that led to the impasse. Instead, it depends on the device to detect many of the action constraint violations, while it uses it knowledge about plans, goals and the situation to detect the other impasses. In this section we will describe how the tutor recognizes impasses of each type.

4.3.1 Action Constraint Violations

The tutor recognizes *action constraint violations* by watching the communications link simulator. Examples of this type of impasse are shown in Figure 3: the NLOAD command was rejected in Case 2 because the NCB-program was in the wrong mode. In addition, Case 2 also shows the NLOAD directive subsequently being rejected because the parameter, MK, specified a non-existent predict set. The simulator is a faithful representation of the link monitor and control system: it rejects commands when system constraints are violated. These constraints constitute a subset of the operator preconditions.

Sometimes a rejection message provides a brief message of explanation with the rejection, but it never provides a means of working around the precondition failure. When the tutor observes that the com-

mand was rejected, it uses this fact to trigger its cognitive model, which determines the reason for the rejection and finds a way of resolving the impasse. The impasse explication process will be discussed in more detail in the next section, but it worth noting that the simulator recognizes the rejection conditions for the input commands and generates the appropriate rejection message. The tutor reads the same rejection message that the student receives, and it determines the reasons for the rejection without consulting the internal mechanisms of the simulator. Thus, recognizing this type of impasse does not depenal on a detailed cognitive model of the student or of the student's plans. Rather, it depends on the simulator to reject inappropriate actions, removing some of the burden of impasse recognition from the tutor and placing it on the simulator. At the same time it does not place the burden of impasse diagnosis and recovery on the simulator, which only needs to recognize whether an action can be taken given the current state of the devices being simulated.

4.3.2 Goal Failure

The idea behind a *goal failure impasse* is that students may rotely follow a procedure without understanding the goals associated with it. A goal failure violation occurs when the student completes a plan (i.e., all of the expected actions have been observed for an active plan) but does not satisfy the plan's goals prior to beginning a successor plan, This type of error may not manifest itself in an obvious form, such as with a directive rejection, therefore, a student may not recognize the impasse immediately. Instead, a goal failure impasse would likely show up later as another type of impasse in a subsequent procedure. Allowed to pass without tutorial intervention, a goal failure impasse could be very difficult for the tutor to diagnose and resolve since the original context of the impasse may have been lost. Our intuition is that it is better to deal with these types of errors as they are recognized by the tutor rather than wait until it becomes an actual impasse at the action or task goal level (i.e., task failure).

Goal failure impasses are recognized using the expectations about actions, plans and goals, generated by situated plan attribution. The tutor is initially cued to this type of impasse when the student takes an action belonging to an inactive plan. If the inactive plan is a successor of an active plan that is complete but whose goals are not satisfied, then the tutor flags the action as a goal failure impasse, which triggers the tutor's cognitive model to diagnose and recover from the impasse in the current situation.

Case 1 in Figure 3 gives an example of a goal failure impasse. The Configure-DSP plan is assumed to be active during this trace. The student issued the SAT 25, which noted earlier should have been SAT 30, so we know that the goal associated with setting the SAT value properly was never satisfied. The Operator issued all of the other Configure-DSP commands (let's assume that the OFST command was properly issued) and moved on to the Coherence-Test plan, which begins with the V NPCG MAN cornmand. The tutor notes that all of the expected commands from Con figure-DSP have been issued and marks the plan COMPLETE. At the same time, it knows that the plan's goals are not satisfied, so it marks the plan UNSATISFIED. Once it observes the NPCG command, it notes that this command belongs to an inactive plan that follows the still active, albeit unsatisfied Configure-DSP plan. The tutor recognizes this situation as a goal failure impasse and decides to intervene. This is a case where the student's error was a result of mis-parameterizing the directive, which can only be detected as a goal failure since it does not violate any action constraints, hence it is detected after the student is apparently finished executing the plan.

4.3.3 Plan Dependency Violations

'l"he reason for classifying certain behaviors as plan dependency violations is to aid the student who is confused about which plans are applicable for a given situation. The TDN describes the precedence constraints among plans. Thus, if the tutor observes the student executing a plan that is clearly inappropriate for the situation, it will intervene with tutorial advice. For example, consider once again Case 1 in Figure 3. l'he student omitted the OFST command and issued the NPCG MAN command, instead. The tutor matches this action with Coherence-Test plan, which is inactive. Clearly, this action was taken at the wrong time. This is a case where the tutor treats the command as a plan dependency im-

passe since resolving the error entails doing the Configure-DSP plan and it is not necessary to have the cognitive model figure this out when it is clearly a violation of the TDN constraints.

Due to the situated nature of tasks in this domain, however, the tutor does not automatically assume a command that does not match an active plan is a TDN constraint violation. The tutor evaluates the command with respect to the command preconditions of and plan goals of active plans. If it appears that the student was attempting to correct an anomaly or an unsatisfied precondition, then the tutor does not intervene. A good example of this is found in Case 3 of Figure 3. In this instance, the student issued the NIDLE REC command at the beginning of the sequence. The tutor recognizes that this command does not belong to the Configure-DSP plan, which is currently active, but that it does satisfy one of the NLOAD preconditions (i.e., it puts the NCB-program into, the IDLE mode.) Consequently, the tutor does not intervene because the command clearly served a purpose for the current situation and the Operator was able to avoid a rejection of the NLOAD command.

This is the weakest category of impasse because it is the most difficult to correctly recognize. It requires being able to understand actions that do not fit into an expected behavior. Our next extension to the tutor will be to recognize and use severe device anomalies or failures to drive the model of expectation and situated plan attribution. The current effort focuses more on normal operations where the range of anomalies can be accommodated by fairly simple reactive plans or knowledge.

5. Impasse Explication

Once an impasse is recognized, the next step is to decide what to say to the student about it. The process of determining what to say is called impasse explication, which has two basic goals: first, explain the nature of the impasse in the context of the current situation, and second, determine how to resolve the impasse so that the tutor can teach the operator the actions required by the situation. Our method for generating explanations for action constraint violations and goal failure impasses uses an executable expert cognitive model of the domain. The model is sensitive to the state of the devices at the point of the impasse; it reactively performs the task, identifies and repairs unsatisfied preconditions while concurrently generating an explanation for the student. Plan failure impasses are treated somewhat differently. Errors of this type do not require a detailed account of how to solve the problem, rather, we consider them to involve mis-executing a default plan.

5.1 Action Constraint Violations

The strategy for generating an explanation for an action constraint violation begins by applying the expert cognitive model at the impasse point; the explanation is constructed as the cognitive model resolves the impasse. We assume that the impasse was caused by a precondition violation that was overlooked by the student. The reason for the oversight could be due to not knowing the precondition, having a misconception about it, or not attending closely enough to the situation, which may have unexpectedly changed. In any of these cases, our cognitive modeling work (described in section 3) predicts that the impasse is a strategic intervention point and that the tutorial content is crucial to helping the student acquire the new skill.

The tutorial content is constructed by first recognizing the situational factors that led to the impasse and then reasoning about the steps necessary to resolve it. The tutor does not simply solve the problem at the impasse point; rather, it internally simulates applying the operator associated with the student's command. The student's command will normally have one or more parameter values (e.g., the NLOAD command requires a parameter value specifying the predicts set, such as JK). The tutor binds the student's command parameter value(s) to the operator precondition variables and proceeds with the task of attempting to apply the operator.

Figure 9 shows a portion of the problem space hierarchy that the Soar-based tutor searches in order to apply the Load-Predicts operator. Each of the preconditions is evaluated in the Verify-Operator-l'reconditions problem space. If a precondition is not satisfied, the tutor searches the Repair-

Unsatisfied-Precondition problem space for a way to resolve the impasse created by the unsatisfied precondition. Explanations arc generated in both of these problem spaces when there is an un.satisfied precondition, detailing the source and nature of the problem and how to resolve it. Sometimes the unsatisfied precondition can be repaired by merely changing the command parameter, while in other cases it may be necessary to select and apply one or more command operators that will change the state of the device that caused the original precondition to be unsatisfied. The tutor internally simulates applying the commands to the devices, and includes these command applications in its explanation to the student.

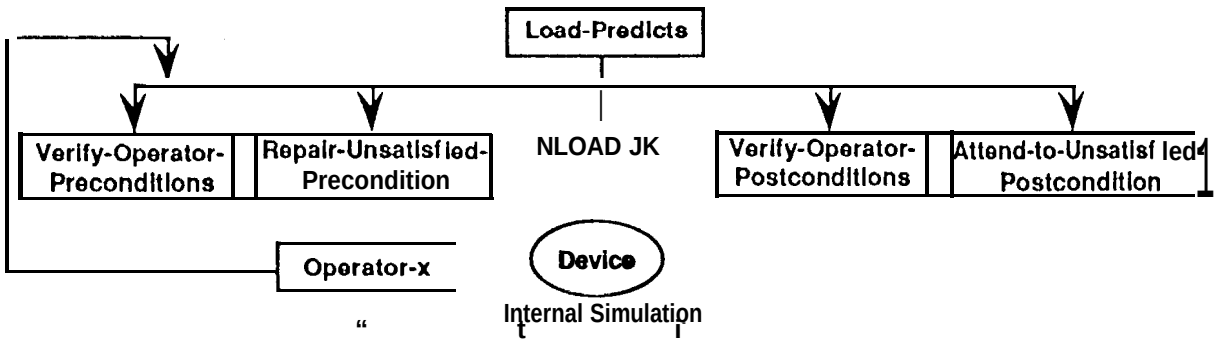

Figure 9 Problem Space Hierarchy of Expert Cognitive Model

To illustrate the explication process we have just summarized, let us return to one of the previous exampies, shown as Case 2 in Figure 3. In this example the student issues the command, V NLOAD JK, and it is rejected, i.e., the simulator issues a message saying, REJECTED. NOT ALLOWED IN NRUN MODE. This interaction is observed by the tutor, and it recognizes the rejection as an action constraint violation. Knowing that the NLOAD command is associated with the Load-Predicts operator, the tutor selects this operator and subgoals into the Load-Predicts problem space.

As previously described, once the Soar-based tutor is in the Load-Predicts problem space, it selects an operator called Verify -Operator-Preconditions which creates another subgoal into the problem space for this operator. In the Verify-Operator-Preconditions space each of the Load-Predicts operator's preconditions (shown in Figure 5) is evaluated with respect to the current situation (i.e., with respect to the state of devices such as those shown in Figure 4.) In this particular case, the precondition named Load-Predicts-PC1 is unsatisfied because the NCB-Program device is in the RUN mode instead of the IDLE mode. Once detected, the tutor adds the information about the unsatisfied precondition to the explanation for the impasse (see Figure 10.1.)

Impasse Explanation			
Device	Attribute	Expected Value	Actual Value
NCB-Program	MODE	IDLE	RUN

Figure 10.1 Explanation of why the NLOAD command failed

Impasse Resolution			
Issue Command:	Affected Device	Affected Attribute	Value
NIDLE REC	NCB-PROGRAM	MODE	IDLE
NLOAD JK	VLBI-PREDICTS	LOADED?	YES
	CON FIG-TABLE	NCOMB	100.0
	...	.....	...
	CONFIG-TABLE	TYPE	DOR

Figure 10.2 Explanation of how to resolve the impasse

The tutor must now find a way of satisfying the Load-Predicts-PC1 precondition, so it selects another operator in the Load-Predicts problem space called Repair-Unsatisfied-Preconditions. Again, the Soar-based tutor subgoals into a problem space for this operator, where it searches for a command operator that will change the NCB-Program from the RUN mode to the IDLE mode. The tutor finds an operator called Idle-NCB-Program which when applied will have the desired effects on the NCB-Program device. The tutor selects the Idle-NCB-Program operator and subgoals into a problem space that contains the same kinds of operators as the Load-Predicts problem space shown in Figure 9, and the process of verifying preconditions for the Idle-NCB-Program is performed. Given that the preconditions for this operator are satisfied, the Soar-based tutor internally simulates issuing the Idle-NCB-Program operator's command, NIDLE REC, and changes its internal model of the NCB-Program device mode from RUN to IDLE. The NIDLE REC command is added to the explanation as shown in Figure 10.2.

Once this is accomplished, the Idle-NCB-Program subgoal terminates, and the tutor continues its problem solving in the Load-Predicts problem space. It begins by r-e-evaluating the previously unsatisfied Load-Predicts-PC1 precondition with respect to the revised device model. Since this precondition is now satisfied (and assuming that all of the other preconditions are also satisfied), the tutor applies the NLOAD command to its internal device model. This command is also added to the explanation for resolving the impasse (see Figure 10.2). The tutor terminates the Load-Predicts operator once it verifies that all of the postconditions have been satisfied, and the explanation is complete.

According to our model of skill acquisition [Hill and Johnson, 19931, the information contained in the explanation is precisely what the student should learn about the situational constraints of the failed command in order to avoid the same impasse in the future. In our modeling work, this new knowledge is initially acquired declaratively, and it is chunked into the form of a procedural skill as the student applies it in a problem solving context. The proceduralization of skill in the model is a direct result of the Soar architecture's learning capability provided by the chunking mechanism [Rosenbloom and Newell, 1986; Laird et al., 1987]. Moreover, our model of skill acquisition closely resembles the account given by Anderson [Anderson, 1983; Anderson, 1989; Anderson et al., 1990].

5.2 Goal Failure Impasse

Goal failure impasses are handled in much the same manner as action constraint violations; the primary difference is the level at which the tutor enters the expert cognitive model's problem space hierarchy. Instead of selecting a command operator, which was the case in example described in the last section, the tutor selects the plan-level operator (see Figure 1) corresponding to the plan whose goals were not satisfied. The tutor's goal in the plan operator's problem space is to select and apply command operators that will satisfy the plan's failed goals.

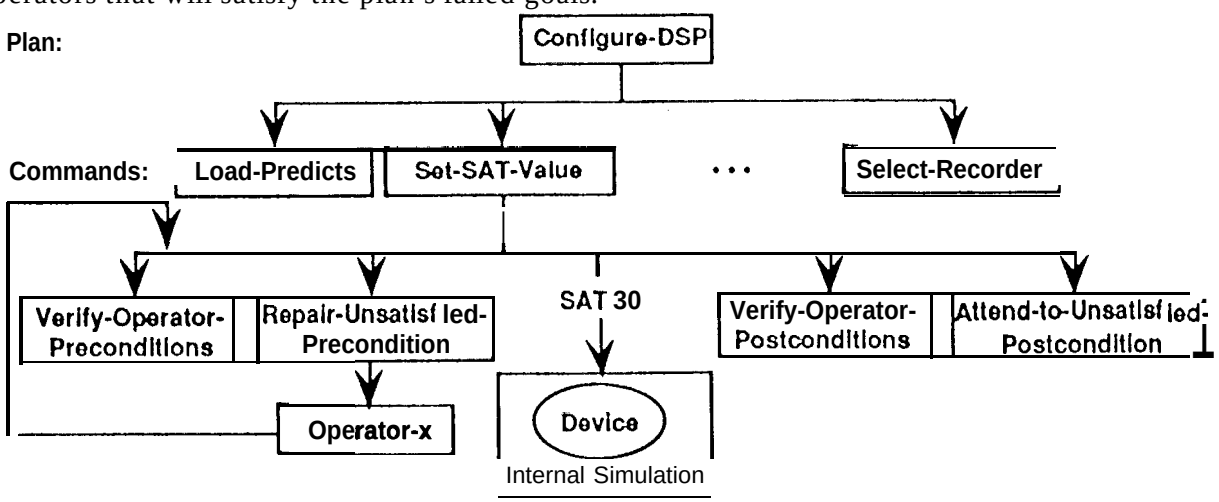

Figure 11 Resolving a goai failure impasse related to the S-band attenuation (SAT) level setting.

A simple example of how the tutor explicates a goal failure impasse is shown in Figure 11. After the student finished the Configure-DSP plan, the tutor notices that the S-Band Attenuation (SAT) value was incorrect y set. The tutor subgoals into the Configure-DSP plan problem spare and resolves the impasse by selecting and applying the Set-SAT-Value operator; this is a simple case where it was only necessary to correct one command of the Configure-DSP plan. One can imagine instances, however, where the plan's unsatisfied goals are not corrected so easily. In such cases, the tutor will search for and apply command operators in the plan's problem space until all of the goals arc satisfied, and the explanation will reflect all of the commands that it selected to satisfy the goals.

5.3 Plan Failure Violations

This type of error appears to originate from a plan-level misconception or knowledge gap rather than at the command level. Either the student does not know which plan applies in the current situation or else does not know the plan's commands. Consequently, the tutor explains plan failure violations in terms of the plans that should be active in the current situation. The active plan's command operators are taught, but the details about their preconditions are omitted. In this way the student first becomes, familiar with the default plan and its commands without being overwhelmed by the situational details of applying the commands.

6. The Tutor Improves over Time

The tutor was implemented to take advantage of Soar's learning capability. The Soar architecture embodies the idea that problem solving is a goal-oriented activity involving the search for and application of operators to a state in order to attain some desired results [Laird et al., 1987]. Search takes place in a hierarchy of problem spaces, where a problem space contains a set of operators and an initial state. The problem space hierarchy is traversed via subgoaling, which takes place whenever the problem solver cannot make any more progress toward a goal in the current problem space. Learning occurs when a subgoal yields a result; the result is stored in a production that summarizes the conditions for subgoaling and the result of the problem space search [Rosenbloom and Newell, 1986]. The chunk can be applied the next time a similar situation arises and the same results can be achieved without searching a subgoal problem space.

There are two consequences of having a tutor that learns: (1) the t utor's performance improves with experience, and (2) the nature of the tutor's problem solving changes as it learns. The first consequence has a bearing on the efficiency of the implementation, As the tutor gains experience with different student impasses, the knowledge of how to recognize and explicate these impasses is chunked into new productions. The chunks improve the tutor's performance significantly, since they limit the amount of search that is required to solve a familiar problem again. The second consequence affects the way we characterize our approach to tutoring, since the tutor's knowledge becomes more procedural as it gains experience.

6.1 Impasse Recognition Chunks

To illustrate how the tutor learns to recognize impasses, consider what happens when the tutor sees an action-response pair (i.e., a student command and device response); it subgoals into a problem space called Analyze- Action-Response, where it searches for a plan that contains the student's command. When the tutor finds a match, it marks the command in the plan as "matched". The tutor then decides whether there is a potential impasse, depending on whether the plan containing the command was active or not and whether the command was accepted or rejected by the device. Two types of potential impasses are identified in this problem space: plan-dependency-violation and action-constraint-violation. The Analyze-Action-Response problem space terminates once the tutor finishes analyzing the action-response pair.

The chunks built while searching in the Am~lyze-Action-Response problem space automatically recognize whether a specific action-response pair is a potential impasse for a given situation. Hence, the

next time that the action-response pair is observed, the tutor will recognize whether there is a potential impasse or not without any further subgoaling or search.

6.2 Impasse Explication Chunks

Chunks arc also built while the expert cognitive model explicates an impasse, Once the tutor has resolved a particular impasse, the resulting explication chunks arc general enough to solve the same problem again even though the parameter values may be different, In addition, there is a transfer of knowledge about how to perform tasks such as verifying preconditions and postconditions, so as new impasses arise some of the previously learned chunks will be applied,

6.3 Improvement

There is a significant improvement in the tutor's performance after it has chunked the problem space hierarchies used to recognize and explicate student impasses. Figure 12 shows the amount of time it takes the tutor to handle action constraint violations for the NLOAD and SAT commands before and after learning.

Command	Recognize Impasse		Explicate Impasse		Total		Ratio: Before/After (Total)
	Before	After	Before	After	Before	After	
NLOAD	0.20	0.08	8.70	0.61	8.90	0.69	13::1
SAT	0.19	0.08	2.70	0.32	2.92	0,40	7::1

Figure 12 Tutor's performance on action constraint impasse before and after learning, measured in seconds

Note that the time it takes to recognize the impasse is constant among these commands, before and after learning; it takes roughly 0.20 seconds to recognize an impasse involving either command before learning, and it improves to 0.08 seconds after learning. On the other hand, the amount of time it takes to explicate an action constraint violation varies, depending on how many preconditions and postconditions must be checked for the command. We chose the NLOAD and SAT commands for this example because they provide upper and lower bounds for this type of impasse. The NLOAD command has the greatest number of preconditions and postconditions and the SAT command has the fewest.

7. Comparison to Model Tracing and Plan Recognition Approaches

We compare our tutoring method to two approaches that have been used in a number of intelligent tutoring systems, namely, model tracing [Anderson, 1990; Reiser et al., 1985; Ward, 1991] and plan recognition [Johnson, 1990; Warriner, 1989; Rickel, 19881. Our tutor resembles elements of both model tracing and plan recognition, though it cannot be completely characterized as one or the other according to current definitions. In fact, we would suggest that the current conception of model tracing should be extended to allow for model tracing at different levels of abstraction. To clarify what this means, consider how model tracing and plan recognition address the tutoring intervention problem.

7.1 The Intervention Decision

Model tracing recognizes errors using an executable performance model of the student to search for production rule paths that account for the student's behavior. If an action can only be accounted for via a mal-rule application, then the tutor concludes that the student made an error and passes this interpretation to the pedagogical model [Anderson et al., 1990]. This differs from our approach in that our tutor does not detect errors by **running a cognitive simulation of the student.** Rather, our tutor focuses on recognizing impasses, and more closely resembles plan recognition than model tracing when deciding whether to intervene.

Whereas model tracing uses procedural knowledge to detect student errors, plan recognition approaches to student analysis typically match the observed behavior to a declarative description of action. This either involves matching and interpreting the student's action sequence using a library of plans [Johnson, 1990; Calistri, 19901, or else interpreting the action with an action grammar or procedure net description of the task [Burton, 1982; Rickel, 1988; Warriner et al., 1990]. In a plan matching approach, errors are detected with real-plans or difference rules, while in the action grammar case an error is any action that can't be parsed by the grammar, or which is only parsed with a model that includes buggy rules. Our method of impasse recognition bears some resemblance to these approaches in that the tutor initially matches student actions to declarative plan descriptions and recognizes potential impasses when actions do not fit the expectations generated by the plans and their goals. It differs in that the tutor does not use manually encoded real-plans or difference rules to recognize the impasse; as the tutor gains experience it effectively generates its own real-plan rules in the form of recognition chunks. Hence, our tutor learns when to intervene. At an abstract level it traces a performance model, but the difference from standard model tracing is that it does not attempt to generate mental states to do so. It traces the student's progress in enough detail to make predictions of what plans the student might be following, and no more.

7.2 Deciding What to Say

One of the strengths of the model tracing is its ability to give a reasoned explanation for an error, once it is detected. In this regard, our tutor resembles a model tracer; it generates explanations using an executable cognitive model that diagnoses possible causes for the error and suggests how to resolve the current impasse. The difference between the two approaches is that model tracing relies partly on its knowledge of real-rules to generate the explanation, while our tutor learns to recognize the causes for errors while applying an ideal performance model to the impasse. The declarative representations of operator preconditions used by the cognitive model end up being proceduralized, to a large extent, as the tutor gains experience. These chunks have the same effect as a real-rule in that they can be used to explicate an impasse.

7.3 Generality of the Approach

The simplicity of our approach is partly due to the constrained nature of the task being tutored. We assume at the outset that the mission that the student is performing is known. This allows us to make predictions of what plans the students are likely to be following. All model tracing and most plan recognition systems make similar assumptions.

However, it should be possible to weaken this assumption and retain the same basic approach. Device failures and anomalies can lead the Operator to carry out different or additional plans. Operators sometimes get confused about which plans arc appropriate for which missions, and carry out inappropriate plans. These cases can simply be treated as alternative sources of expectations for plans. Tutorial intervention will continue to focus on the appropriateness of attributed plans to the situation at hand.

8. Results

We have made a number of claims about how students learn, and based on this, how an intelligent tutor can effectively teach in a monitor and control domain. In order to assess the veracity of these claims, we plan to evaluate the tutoring system in two ways: first, by testing it with novice operators, and second, by soliciting the reactions of expert operators and system engineers to the method and accuracy of the advice given by the tutor.

The tests involving novice operators will divide the participants into two groups: a control group and a test group. Both groups will be given identical tasks to perform on the link monitor and control training simulator, and the simulator devices will also start in the same state for both groups. The primary difference between the two groups will be that the test group will receive advice from the tutor, while the

control group will not. We will measure the task performance of each group over a number of trials on the same task; data will be collected on how the student performed on each trial in terms of (1) time elapsed, (2) number of commands needed to complete the task, and (3) the number of impasses or errors committal. This data will help us make a formative evaluation of how helpful the tutor is with respect to just using the simulator.

9. Conclusions

We have introduced a new approach to tutoring that focuses on recognizing student impasses through the use of situated plan attribution. Unlike plan recognition, we assume that behavior is situated rather than plan-based and therefore cannot necessarily be recognized as plans. Plans serve as resources for action which must ultimately be situated in the world. Likewise, we do not attempt to understand every student action as is the case with both model tracing and plan recognition. Instead, our tutor focuses on recognizing impasse situation% From the tutor's perspective, the task of recognizing certain impasses is simplified by listening to the device (i.e., directive rejections) rather than trying to generatively recognize the action as an error. Once an impasse is recognized, our expert cognitive model explicates the situation at the impasse point, thereby incorporating one of the strengths of model tracing.

10. References

[Anderson, 1983] John R Anderson. The architecture of cognition. Harvard University Press, 1983.

[Anderson, 1988] John R. Anderson. "The Expert Module". Foundations of Intelligent Tutoring Systems, edited by Martha C. Poison and J. Jeffrey Richardson. Lawrence Erlbaum Associates, Inc, 1988.

[Anderson, 1989] John R. Anderson, "Use of analogy in a production system architecture," in Similarity and Analogical Reasoning, edited by Stella Vosniadou and Andrew Ortony. Cambridge University Press, New York, 1989.

[Anderson et al., 1990] John R. Anderson, C. Franklin Boyle, Albert T. Corbett and Matthew W. Lewis. "Cognitive modeling and intelligent tutoring," Artificial Intelligence, 42, pp 7-49, 1990.

[Brown and VanLehn, 1980] John Seely Brown and Kurt VanLehn. "Repair theory: a generative theory of bugs in procedural skills," Cognitive Science (4), pp. 379-426,1980.

[Burton, 1982] Richard R. Burton. "Diagnosing bugs in a simple procedural skill," Intelligent Tutoring Systems, edited by D. Sleeman and J.S. Brown. Academic Press, Inc., 1982.

[Calistri, 1990] Randall J. Calistri. "Classifying and detecting plan-based misconceptions for robust plan recognition." Ph.D. Dissertation, Technical Report No. CS-90-11, Department of Computer Science, Brown University.

[Chen et al., 1991] Thomas T. Chen and Diann Barbee, "Integrating an intelligent tutoring system with an existing simulator, " Proceeding of the 1991 Conference on Intelligent Computer-Aided Training, NASA/Johnson Space Center, Houston, Texas, November 20-22,1991.

[Fayyad and Cooper, 1992] Kristina Fayyad and Lynne Cooper, "Representing Operations Procedures Using Temporal Dependency Networks," Proceedings of the Second international Symposium on Ground Data Systems for Space Mission Operations, SPACEOPS-92, Pasadena, CA, 16-20 November, 1992,

[Galdes, 1990] Deborah K. Galdes. An empirical study of human tutors: the implications for intelligent tutoring systems. Ph.D. Dissertation, The Ohio State University, 1990. Order Number 9031068, UMI Dissertation Services, Ann Arbor, Michigan.

[Hill and Johnson, 1992] Randall W. Hill, Jr. and W. Lewis Johnson. "Designing an intelligent tutoring system based on a reactive model of skill acquisition," World Conference on Artificial Intelligence in Education (AI-ED 93), Edinburgh, Scotland, 1993.

[Hill and Lee, 19921 Randall W. Hill, Jr. and Lorrine Lee, "Situation Management in the Link Monitor and Control Operator Assistant," Proceedings of the Second International Symposium on Ground Data Systems for Space Mission Operations, SPACEOPS-92, Pasadena, CA, 16-20 November, 1992.

[Johnson, 1990] W. Lewis Johnson. Understanding and Debugging Novice Programs. Artificial Intelligence, 42, pp. 51-97, 1990.

[Laird et al., 1987] John E. Laird, Allen Newell and Paul S. Rosenbloom. "Soar: An architecture for general intelligence," Artificial Intelligence, 33(3), 1987.

[Newell, 1990] Allen Newell. Unified Theories of Cognition. Harvard University Press, 1990.

[Reiser et al., 1985] Brian J. Reiser, John R. Anderson and Robert G. Farrell. "Dynamic student modelling in an intelligent tutor for lisp programming," Proceedings of IJCAI-85, Los Angeles, CA, 1985.

[Rickel, 1988] Jeff Rickel. "An intelligent tutoring framework for task-oriented domains," Proceedings of the International Conference on Intelligent Tutoring Systems, Montreal, June 1-3, 1988.

[Rosenbloom and Newell, 1986] Paul S. Rosenbloom and Allen Newell. "The chunking of goal hierarchies: a generalize model of practice," Machine Learning, Volume II, pp. 247-288, edited by Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1986.

[Suchman, 1987] Lucy A. Suchman. Plans and situated actions. Cambridge University Press, New York, 1987.

[Ward, 1991] Blake Ward. E'I'-Soar: Toward an ITS for Theory-Based Representations. Ph.D. Dissertation, CMU-CS-91-146, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[Warinner et al., 1990] Andrew Warinner, Diann Barbee, Larry Brandt, Tom Chen, and John Maguire. "Building an intelligent tutoring system for procedural domains," Proceedings of the First CLIPS Conference, pages 881-892, NASA/Johnson Space Center, Houston, TX, 1990.

11. Acknowledgements

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Dr. Johnson was supported in part by the Advanced Research Projects Agency under contract number NOO013-92-K-2015. Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of the U.S. Government or any agency thereof.

We thank Paul Rosenbloom and the members of the Soar group at USC-ISI for their helpful comments on this paper. We also thank Lynne Cooper, Lorrine Lee and members of the Link Monitor and Control Operator Assistant team at JPL for their helpful input.