Building a Computer Tutor: Design Issues

Beverly Woolf
David D. McDonald

COINS Technical Report 87-77

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

Because no two students answer questions in the same way, an effective computerized tutor must customize its response to the individual student.

Building a Computer Tutor: Design Issues

Beverly Woolf and David D. McDonald

University of Massachusetts

An effective tutor must deal with a fundamental problem of communication: to determine how messages are received and understood, and to formulate appropriate answers. A human tutor, therefore, takes the time to double-check and review a student's knowledge to find out whether or not he understands what the tutor has said. Compounding the situation is the fact that a student is generally unaware of what he does not know. This means that a tutor, more than a typical speaker, must verify that both parties know what information has been covered, what is missing, and which communication might be erroneous.

In this article we discuss how a deep understanding of a student can be constructed in an artificial intelligence program and how this understanding, coupled with a facility for language generation, can be used to build a flexible machine tutor.

The nature of tutoring

Tutoring is a linguistic exchange whose goal, in general, is to clarify a body of knowledge to which the student has already been exposed—for example, knowledge obtained through lectures or reading. It involves directing a dialog so that the responses remain appropriate even in the face of errors. Not every activity we might normally associate with "tutoring" is, in fact, an effective tutoring tool. Classroom activities such as pretests, posttests, and drill-and-practice exercises are a good case in point; they have developed over time in the context of classroom teaching because the ratio of students to teachers is large and one-on-one "sensitive" tutoring has not been possible. In short, these activities do not represent the "essence" of tutoring. One of our purposes in developing competent machine tutors is to reduce the large student-to-teacher ratio and to establish one-on-one interaction as the norm, not the exception.

To give you an idea of what an effective tutor does, we have borrowed this example (Example 1) from Stevens[1] of an exchange between an expert human tutor working with a student on understanding rainfall:

Tutor:	Do you know what the climate is like in Washington and Oregon?
Student:	Is it cold?
Tutor:	No, it is rather mild. Can you guess about the rainfall there?
Student:	Normal, I guess.
Tutor:	Well, the Japan Current, which starts in the Southeastern Pacific, goes along the coast of Japan and across the North Pacific, ending up off the coast of Washington and Oregon. How do you think that current affects the climate there?
Student:	It's probably rainy.
Tutor:	It is indeed very wet; there are rain forests in both Washington and Oregon. What does that tell you about the temperature of the Japan Current?
Student:	It's warm. (1)

The protocol of the exchange is taken from an earlier investigation of human tutoring behavior[1] and shows how the expert remains responsive and sensitive to the student's knowledge level. We suggest that the tutor began to question the student about general topics, in this case "climate" and "rainfall," in an attempt to assess his "frontier" of knowledge. Since the student answered the first two questions incorrectly, we feel that the tutor decided to change his strategy and therefore elected to provide the student with additional information from which he might be able to infer the correct information. The strategy worked—the student answered the next two questions about "rainfall" and the "temperature of the Japan Current" correctly.

The tip of the iceberg in discourse. Designing a machine tutor that can display the amount of flexibility shown by the example above presents some very interesting problems. For example, how is the machine going to recognize a speaker's unspoken problems or intentions? We call this the "tip of the iceberg" problem, by which we mean that the words spoken by the student or the teacher are only a small portion of what they know about each other and what they are communicating in the dialog. This problem has been analyzed by Allen, who built a system to model the knowledge needed by an effective agent at a train station.[2] The discourse we have created is in the style of Allen's examples. It is predicated on the agent anticipating or predicting the traveler's problem and on handling this unspoken problem:

Traveler: Is this the gate for the train to New York?

Agent: No, you want gate number 44 and the train leaves at 6:33." (2)

If the answer to the traveler's question had been "yes," that by itself would have been enough and would have solved the traveler's problem. Since the answer was "no," however, a simple "no" by itself *would not* be a sufficient answer as it would not have addressed the traveler's actual problem. By giving the additional information, the agent showed that he recognized that what the traveler was *saying* was only a small part—only the tip of the iceberg—of what he actually intended to communicate.

The "tip of the iceberg problem" in tutoring is to recognize the student's unspoken confusions and misconceptions. By maintaining a record of the student's previous errors or by directly questioning him about his misconceptions, a machine tutor can acquire the same kind of information used above by our train agent. Yet, even with such information, a machine model of the student can never be entirely accurate and must be continually modified and updated.

Earlier tutoring systems

Below we describe several tutoring systems* organized around the four types of knowledge that a tutor—human or computer—must have in order to teach effectively: the subject area, the student's information, how to teach, and how to communicate. Though a tutoring system must handle each of these aspects of tutoring, it need not allocate a separate module to each aspect. For purposes of this article, however, we shall refer to each type of knowledge as though it had been built into its own module and shall describe systems in which one module was, at least in part, the focus of the research project.

*Many of the early tutoring systems were built as laboratory experiments and few have been used extensively with students.

Part 1: the expert module. The first module in any tutoring system contains knowledge of the subject area and covers the possible rationales for the student's answers. It is used to translate the student's input into a usable form—one that can be evaluated against the expert's knowledge. In some cases, this module provides problems to solve or topics to discuss. It may not be able to solve the problems it presents to the student, but it must at least recognize an incorrect solution. This component was first addressed by the artificial intelligence community early in its efforts to build intelligence into tutoring systems. When heuristics began to be included in this module, tutoring systems began to respond to a broader range of input and, in some cases, to understand and generate natural language. Intelligent heuristics in this module gave it more flexibility to interpret the answers of a nontraditional student—that is, the student who has the correct answer, but expresses it in a fundamentally different way. These heuristics also enhanced the system's ability to express its own knowledge, enabling it, for example, to explain its problem-solving reasoning.

Currently, there are two outstanding examples of systems that incorporated intelligence into the expert module: Sophie[3] and Guidon.[4] Sophie, the Sophisticated Instructional Environment, is a landmark effort in the development of languages for expert modules. It is designed for electronic troubleshooting and allows a student to test and debug a hypothetical malfunction in a simulated electrical circuit. The tutor evaluates a student's hypotheses about the circuit failure, considers the data available to him, and advises him about his analysis, often proposing its own fixes. The tutor also refuses to carry out any probes or to answer any queries that it does not consider appropriate for solving the problem. The following (Example 3) is a sample interaction with the Sophie system. The circuit components referred to in the example (such as, resistor 8 [R8], node connection 1 [N1], and transistor [Q1]) have been presented to the student in a schematic diagram of the circuits.

Welcome to Sophie, a reactive learning environment.
The piece of equipment you are studying is a regulated power supply.
Would you like to start with a faulted instrument that needs fixing?

>> *Yes*

The instrument has a fault and its present control settings are

CC	1.0	CURRENT CONTROL
CR	HIGH	CURRENT RANGE SWITCH
LOAD	2.0	LOAD RESISTANCE IN OHMS
VC	1.0	VOLTAGE CONTROL
VR	HIGH	HIGH VOLTAGE RANGE SWITCH

> > *What is the output?*

The output voltage is 11.7 volts.

> > *What is it in a working instrument?*

In a working circuit the output voltage is 19.9 volts.

> > *What is the voltage between R8 and Q1?*

The voltage at N1 is 47.6 volts. (3)

Sophie can perform sophisticated reasoning when it examines a student's inquiries. For instance, it can answer a question like "If the base-emitter junction of the voltage-limiting transistor opens, then what happens to the output voltage?" It does this through two mechanisms: a "semantic" natural language facility and a general-purpose simulator. The natural language facility uses a *semantic grammar* that replaces the usual syntactic categories, such as noun, verb, and adjective, with semantically meaningful categories, such as transistors, resistors, and voltages. For each concept there is a grammar rule that gives alternative ways of expressing or understanding that concept. The general-purpose simulator interprets and applies the modifications suggested by the student to the existing simulation of the circuit. The result of the final simulation is the production of a voltage table that represents voltages at each terminal in the modified circuit. The student's question is then answered in terms of the hypothesized voltages.

A second example of a system, in which intelligence is built into the expert module, is the Guidon system.[4] Guidon helped demonstrate the need for a cognitive base to the expert module and defined the importance of placing logical and relational abstractions, both of which are important in learning and tutoring, into the expert knowledge base. The goal of Guidon is to tutor the knowledge from an expert system. To date, its primary focus has been Mycin, a medical expert system that suggests treatment for infectious diseases. The example below (Example 4) is of an interaction with the Guidon system. In it, the medical student chooses from a menu of over 30 options and, for instance, can request data (for example, results of lab tests), ask for an evaluation of the medical situation, question the rule being discussed, request assistance, or suggest an hypothesis.

> > *Factors*
; *The student asks for the factors or subgoals*
; *of the rule being used to determine the*
; *nature of the infection.*

The following factors will be useful to determine the type of infection:

3a. whether the meningitis is partially treated
3b. whether a lumbar puncture has been performed on Patient 538
3c. the duration of the neurological signs
3d. the CSF protein
3e. whether Patient 538 has a rash consistent with Herpes Zoster

Based on what we have already mentioned, you should be able to conclude about 3b.
; *Guidon keeps track of what has been discussed*
; *and conjectures about what the student should know*

> > *Use 3c*

The duration of the neurological signs is 7.2 hours. Using other information available to you now, the duration of the neurological signs enables you to conclude that the type of the infection is bacterial viral.

; *The student asked how the datum in 3c can be*
; *used in this case. He is given information specific*
; *to this case and then a summary of its use as a part*
; *of the rule under consideration.*

> > I know
; *The student indicates that he has determined*
; *a particular subgoal. In this example Guidon*
; *will supply the correct subgoal.*

Good. Mycin has also made a decision.

The site of Culture-1 is evidence that a lumbar puncture has been performed on Patient 538. (4)

Guidon was originally implemented by "reversing" the 450 rules of Mycin, its expert system. This implementation was ineffectual, its author said, largely because medical diagnosis is not made "cookbook" style—that is, medical practitioners do not diagnose diseases by using perfect recall on hundreds of medical facts and rules, nor do they use rules such as those originally implemented in the Mycin system. Subsequent psychological research into medical problem solving suggests that the rules used by practitioners to diagnose diseases are embellished with knowledge about causal reasoning and cross-referencing, which, for instance, causes a rule to be brought to mind at the appropriate time.

The suggestion has been made that the original rules of the Mycin System represent "compiled" knowledge devoid of the low-level detail and relations necessary for learning or tutoring. In order to use these same rules for tutoring, Guidon would have to "decompile" and augment them with the data and diagnostic hypotheses that the medical practitioner uses implicitly. The resulting "augmented" teaching system required, for instance, procedural knowledge about how to use the rules for problem solving—for example, users were sometimes advised to try searching the rules by using top-down refinement. It also included aspects of the rules by which the problem-solving strategies were to be brought to the student's mind at the appropriate time, as well as took into account rules that helped the student to remember a particular rule and to focus on one set of "associated" rules over another. The original list of 450 rules was amended; to include such relations between rules, and a newer version of Guidon, one based on these kinds of design changes, has had increased success as a tutoring system.

Part 2: the student model. The second part of a tutoring system contains information about the individual student. This is used to predict the student's level of understanding or to recognize his particular learning style. For example, a student may use an uncommon, though not less valid, set of rules and processes in solving a problem and the tutoring system ought to recognize this kind of learning and find a way to work with it. For instance, the system might provide extensive questions to identify and track the anomalies in the student's knowledge or thought processes.

Early systems had almost no student model. At best, they used a stereotypical representation of the knowledge domain that was tagged according to topics that were presumed to be "known" or "unknown" by the student. This type of system was later called an *overlay modeler.* Other modelers have emerged. For instance, a *skill modeler* is an overlay model in which knowledge of the expert is represented as skills and the student is tested on whether he has mastered the set of skills. A *bug modeler* encodes information about bugs and misconceptions. It may, or may not, be an overlay modeler. Two systems exemplify the use of AI techniques to build intelligence into the student model: West,[5] a skill modeler, and Buggy,[6] a bug modeler.

West is a coaching environment for a Plato game. It evaluates and suggests improvements to a student's skill in using elementary-level arithmetic constructs like parentheses and exponents. The object of the game is for the student to move a player across an electronic game board a distance determined by the solution to an algebraic expression that the student must write and then solve. At each move, the student's skill in writing the algebraic equation is compared to the expert's solution for the same skill, and, if the two solutions differ, a friendly coach can intervene and provide the student with helpful hints about how to improve his game or make better moves.

Buggy is an intelligent student modeler without being an overlay modeler. It models the bugs made by students solving simple arithmetic problems and presents computational evidence that such errors result from explicit and systematic deviations from correct procedures. The bizarre nature of arithmetic bugs, as represented in the Buggy system, suggests that no *subset* of expert knowledge would ever be appropriate as a student model because no subset would contain the *incorrect* procedures used by novice students. In addition, Buggy demonstrates that incorrect student behavior is not a modification nor a simplification of expert behavior—previously a commonly held belief among authors of tutoring systems. It proposes, instead, that errors result from *correctly* following *incorrect* procedures rather than from *incorrectly* following *correct* procedures.

The Buggy system is designed to give a teacher experience in diagnosing incorrect arithmetic procedures and practice in discovering the underlying causes of student errors. The following (Example 5) is a sample of an interaction with Buggy:

```
Welcome to Bugland. I have chosen a bug.
Here is an example of the bug.
    17
  + 5
    --
    13
Now you can give me problems to determine the bug.
    ; BUGGY asks the teacher to suggest problems
    ; for it to solve in order to deduce the bug.
>>     51     99     68
>>  +1707    +99    + 9
       21     36     23
    ; The teacher enters the numbers to add and
    ; BUGGY gives the answers that correspond to the bug.
>> Got the Bug!
Please describe the bug you think I have:
>> The student is adding all the digits together.
    ; BUGGY does not "read" the description of the bug
    ; because it does not have natural language understanding
    ; abilities. Instead it generates additional
    ; problems for the team to solve.
Here are some problems to test your theory about the
bug.
What are:
          33   1091     8     28    90
         +99    +60   +34    +70    +6
>>        24     17    15     17    15
    ; Now the teacher gives Buggy the answer to the problems
    ; posed by the system.
Very good. My description of the bug is: The student
always sums up all the digits, with no regard to col-
umns.                                                (5)
```

Buggy can reproduce several hundred "bugs" that might appear in a student's solution of an addition or subtraction problem. The system represents human arithmetic skills, such as "add 1 to the next column," as procedures linked into a network. If the procedure specified by the student is correct, then Buggy performs the addition or subtraction problem correctly. If the procedure is incorrect, the solution contains systematic errors of the kind shown in Example 5.

Part 3: the tutoring component. The third part of a tutoring system contains the strategies, rules, and processes that govern the system's interactions with the student. It includes, for instance, *how* to tutor, *what* instructional tools to try, and *why* and *how often* to interrupt the student. Some of the reasonableness or intrusiveness of a system is determined by this knowledge. This part of a teaching system is applied after the expert and student modules have been accessed and some assessment made about the level of the student's knowledge.

For instance, a reasonable teaching rule is to correct a student when he makes a mistake. However, when this mistake follows a series of wrong answers about the same subject and when the student has demonstrated weakness in the subject area, we believe that a more appropriate response might be to briefly acknowledge the incorrect response and to move on to new data that might supplement the student's knowledge and help him answer the initial question.

This part of the tutoring system is not responsible for text generation, discourse management, or input/output behavior of the machine. These activities, required of any interactive discourse system, rightly belong in their own fourth component, called the *communication module* (see below), which determines the syntactic and rhetorical features of the discourse. The tutoring module, on the basis of the tutoring objectives of the system, handles only how to respond; it makes decisions about which material to present and which questions or examples to suggest, but should leave questions of form and low-level coherency to the communications module.

Part 4: the communication module. The communication module is limited to generating grammatically correct and rhetorically effective output to (1) help the tutoring component do its job and (2) to interpret the student's responses in terms of the categorizations that the tutoring component is sensitive to. Historically, the communication module has been the last to receive the attention of AI researchers, and today it reflects the fewest additions of heuristic knowledge and techniques from the AI field (with one notable exception mentioned below).

An effective tutoring system demands an intelligent communication module. Once we learn how to *ask* the right questions and how to *focus* on the appropriate issues, we will have come a long way toward building a tutor that acts as a partner, rather than a ruler. As we have said, effective communication does not mean natural-language understanding or generation; natural-language processing has been achieved to some degree. Rather, effective communication requires looking beyond the words that are spoken and determining what the tutor and student *should* be communicating about. This problem becomes particularly acute when the student organizes and talks about knowledge in a different way from the way the expert does it.

As we said, few systems have developed intelligence in the communication module. Guidon is an exception; it carries on a flexible dialog with the student because it uses inference techniques to "tease apart" the student's knowledge. It also selects which dialog is most appropriate to use in a given situation and makes the selection based on the tutor's inferences about the previous interactions and the student's current information. Guidon can switch its discussion to any point based on an AND/OR graph, which represents the rules of the expert system. It can also respond to a student's hypothesis using a variety of techniques—such as "entrapment," which forces the student to make a choice leading to incorrect conclusions—as a means of revealing some aspect of his (mis)understanding.

Discourses produced by Meno-tutor

Meno-tutor is an example of a machine tutor that uses intelligence within the tutoring components. Currently under development at the University of Massachusetts,[7] Meno-tutor has the ability to examine earlier discourses with a student and adapt its discourse appropriately; for instance, it will engage the knowledgeable student in a way that is fundamentally different from the way it engages the confused one. We call this kind of system "context-dependent" and contrast it with what we call "retrieval-oriented" systems, such as Sophie[3] and West.[5] Note that while we have emphasized guiding the learner based on what the tutor knows about him, other systems have placed their emphasis on retrieving the correct answer. Our approach has been to consider the most flexible and most appropriate response, given the context. The other systems often had the goal of retrieving the correct answer, with the I/O routine acting merely as a front end to a knowledge retrieval system.

Meno-tutor actually produced the text in the first human tutoring discourse of this article (Example 1); that is, a computer tutor was built that reproduced the discourse in Example 1. This was done by analyzing the high-level transitions and common speech patterns of this and 12 other human dialogs. The system was then "reverse engineered"; that is, it was given the structures and knowledge necessary to make the same transitions and to produce the same text.[8] For instance, the tutoring system in Example 1 recognized that the student made two wrong answers, and it inferred that his knowledge was limited.* It then "decided" that the question/answer approach, which had been used until then, was ineffective and should be changed. It also judged that a new topic, the "Japan Current," should be discussed because it is a dominant influence behind the region's climate. The system decided to supply the additional data in a descriptive rather than interrogative style because the student seemed confused and might profit from the addition of supplemental data. This first discourse was generated by Meno-tutor in such a way that all the decisions, topics, and tutoring styles available to the system were not only explicit to us (discussed later), but also that we could modify them. This flexibility allowed us to generate additional discourses.

But we have moved beyond the reverse-engineering in this first discourse and have begun to explore the "tutoring space" our apparatus defines by varying both the domain and the particulars of the rules. The discourse in Example 6, for instance, is based on the same domain as the first discourse, but is done in an alternative tutoring style—one brought about by modifying the "meta-rules" that govern whether the tutor explores the student's frontier (Example 1) or probes the student's misconceptions about the current topic as soon as the first mistake is made (Example 6):

Tutor:	Do you think it rains much in Oregon?
Student:	No.
Tutor:	Why do you think it doesn't rain much in Oregon?
Student:	Because the neighboring states are dry.
Tutor:	The neighboring states are dry, but you can not generalize from that. In fact, it does rain a lot in Oregon. Can you guess what causes the rain there?
Student:	Well, there are high mountains in the region and the states border the Pacific Ocean.
Tutor:	Yes. The Pacific borders Oregon. How do you think it is involved in the heavy rainfall there?
Student:	Ocean air rises and cools upon reaching the land.
Tutor:	That is right. (6)

Two meta-rules from Example 1 were modified to achieve the discourse in Example 6. The first rule, as modified, caused the tutor to change its tutoring strategy. In the earlier discourse, this rule was used conservatively; the transition was made only after several topics were com-

*It's not that those answers were simply "wrong," rather that they reflected reasonable default assumptions about the weather in "northern states." An attempt to probe the student's default assumptions is made in the next discourse (Example 6).

pletely discussed and the tutor had some confidence about the student's knowledge (or lack of it). In this discourse, however, the modified rule was applied after a single incorrect answer, thus shifting the focus of the discourse abruptly at its very beginning.

The second modified rule caused the tutor to question the student about misconceptions. Typically, this rule is triggered after all topics are complete, either by the questions about them having been answered correctly or by the student having been corrected by the tutor. In the second discourse (Example 6), however, the rule was modified to eliminate that requirement. The effect was that the rule was activated after the student's first incorrect answer that the tutor recognized as being linked to a common misconception.

In our second exploration of the tutoring space, we substituted a new knowledge base for the facts about rainfall—namely, a knowledge founded on base elementary Pascal looping concepts. The focus of this Pascal tutor is on the misconceptions behind the student's explicit programming errors (see Examples 7 and 8). The model for the misconceptions drew on the results of extensive cognitive studies about how novices learn Pascal constructs.[9] These cognitive studies, and Meno-tutor itself, were part of a larger research effort directed at building an on-line, runtime support system for novice Pascal users.[10] As a part of this effort, the "Bug Finder," was also developed and used for four semesters in classes of several hundred students at the University of Massachusetts. The Bug Finder detected runtime semantic errors in Pascal programs and passed messages on to the tutor about the location of the error, the names of variables associated with the error, etc. The Bug Finder has since been retired, and the discourse in Example 8 below was generated based on simulated messages from the Bug Finder and simulated student input.

This program (Example 7)

```
1      PROGRAM LESSON1(INPUT, OUTPUT);
2         VAR
3            SUM,GRADES,STUDENTS:INTEGER;
4            MEDIAN:REAL;
5         BEGIN
6            SUM: = 0;
7            STUDENTS: = 0;
8            READ(GRADES);
9            WHILE GRADES > 0 DO
10              BEGIN
11              SUM: = SUM + GRADES;
12              STUDENTS: = STUDENTS + 1;
13              GRADES: = GRADES + 1
                ↑(should be READ(GRADES))
14              END;
15           MEDIAN: = SUM/STUDENTS;
16           WRITELN
17             (THE MEDIAN GRADE IS, ME-
                DIAN:8:3)
18        END.                                    (7)
```

was submitted by a novice programmer, and the dialog it engendered from Meno-tutor is reproduced below in Example 8:

Tutor: Do you know that GRADES, in line 9, is a control variable for the WHILE loop?
Student: Yes
Tutor: Does that mean that GRADES should be
a) Initiated before the loop?
b) Added into SUM within the loop?
c) Modified within the loop?
d) Both a) and c)
Student: C
Tutor: You seem to understand about the control variable of the WHILE loop. Now let's talk about some other command statements.
What do you think READ (GRADES) in line 8 does with the value of GRADES?
a) Reads all values of GRADES at one time.
b) Reads in a single value of GRADES.
c) Assigns several new input values to GRADES.
Student: A
Tutor: Then do you think that GRADES: = = GRADES + 1 updates the next of those values?
Student: Yes
Tutor: There is your misconception.
Adding one to a variable is a common way to step through a range of values. But it cannot apply here because your READ instruction in line 8 retrieves only a single value. You should replace line 13 with a second call to READ (GRADES). (8)

Given the program in Example 7 and the parts of the WHILE loop that were programmed correctly, the tutor inferred that the student understood the basics of loop programming. To be certain that the tutor and the student shared a common vocabulary, the tutor asked two questions; both were answered correctly, suggesting that the student understood the rudiments of looping constructs and the role of the control variable. In the third question, the Meno-tutor analyzed the student's grasp of deeper programming concepts. Prior studies[9] have linked several explicit programming errors with conceptual misunderstandings of looping constructs. For instance, the programming error in this student's program has been linked with a belief that either (1) a single call to READ will access all the values of a variable at once or (2) that updating the control variable (line 13) will do a tacit READ through each value of the variable each time through the loop. The tutor's third and fourth questions discriminate between these two misconceptions and indicate, as is often the case, that the student had both misconceptions (both answers were wrong). The tutor's last response is an example of the "grain of truth" speech pattern—it confirms and reinforces the student's correct knowledge about programming while pointing out the inappropriate components of that answer.

The changes required to produce each discourse are described in Woolf.[8] Though the number of discourses produced is still small, the fact that our architecture allowed us to produce varied, but still quite reasonable, discourse as we changed the particulars of the rules, substantiates the overall effectiveness of our design.

Meno-tutor architecture

Meno-tutor provides a general framework within which tutoring rules can be defined and tested. It is not a fully capable tutor for any one subject, but rather a vehicle for experimenting with tutoring in several domains. At this point, its knowledge of the two domains on which it has been defined is shallow.* However, the mechanism for managing student-tutor interactions is generalizable and applicable to new domains—that is, tutoring rules and strategies can be held constant while the knowledge base is altered to effect a change in domain of discourse. Meno-tutor is thus a generic tutor, but is not committed by design to a single subject.

Meno-tutor separates the planning and the generation of a tutorial discourse into two distinct components: the tutoring component and the surface language generator. The tutoring component makes decisions about what discourse transitions to make and what information to convey or query; the surface language generator takes conceptual specifications from the tutoring component and produces the natural-language output. These two components interface at the third level of the tutoring component as described below. The knowledge base for the tutor is a KL-One network, annotated with pedagogical information about the relative importance of each topic in the domain.

The tutoring component is best described as a set of decision units organized into three planning levels that successively refine the actions of the tutor (Figure 1.) We refer to the network that structures these decisions, defining the default and meta-level transitions between them, as a discourse management network, or DMN. The refinement at each level maintains the constraints dictated by the

*Meno-tutor had been developed without a full-scale natural language generator or a means to fully understand natural-language. The conceptual equivalent of a student's input is fed by hand to the tutor (that is, what would have been the output of a natural-language comprehension system), and the output is produced by standard incremental replacement techniques. We have not yet worked with Mumble,[11] our surface language generator, because we haven't yet invested in building a large enough knowledge base that could then be translated into portions of Mumble's dictionary. Our intent is to develop a complex knowledge base, possibly in the domain of Pascal, to extend the surface-language generator to deal with the domain and to build a simple natural-language parser to interface with the student.

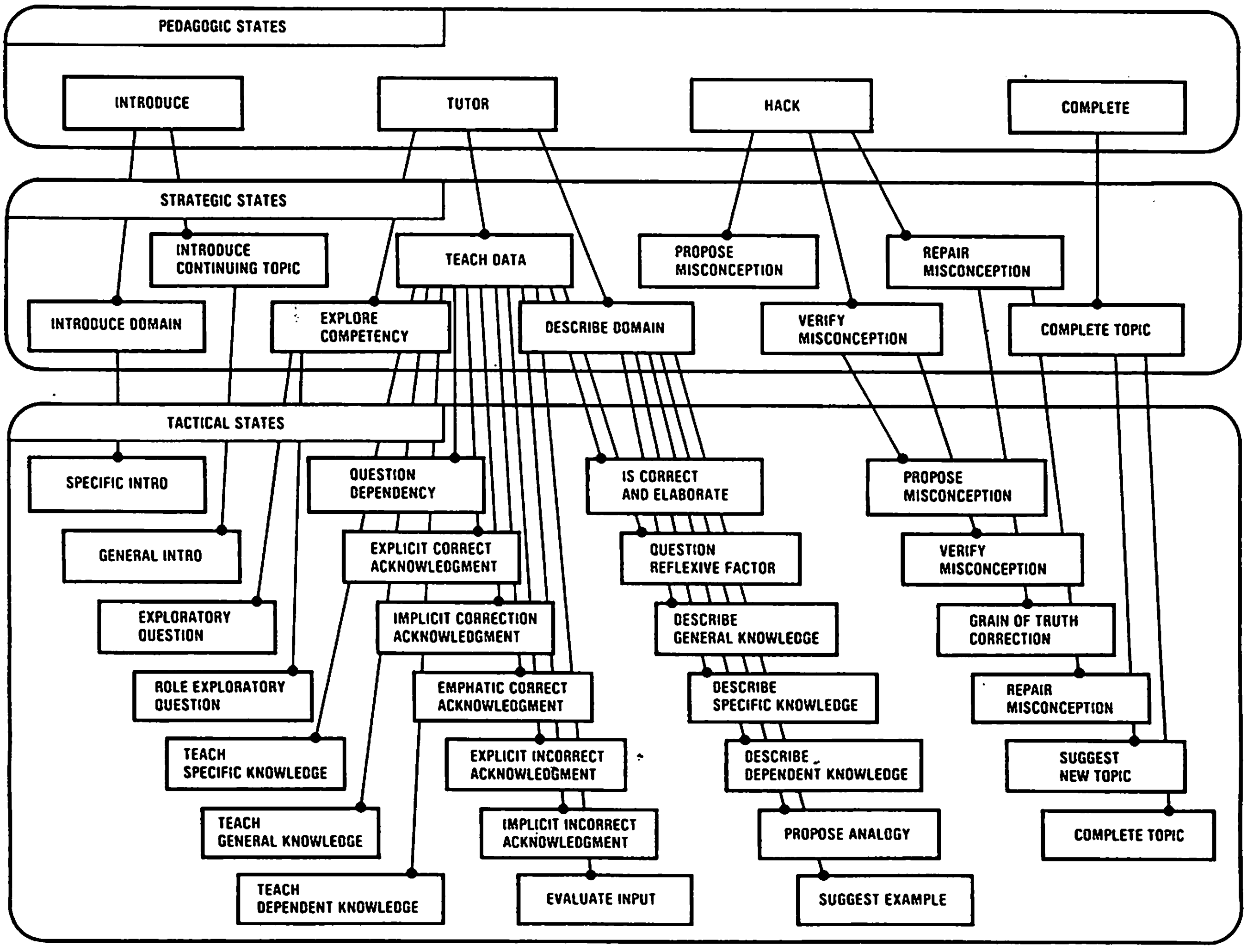

Figure 1. The discourse management network used by the tutoring component.

previous level and further elaborates the possibilities for the system's response.

At the highest level, the discourse is constrained to a specific tutoring approach that determines, for instance, how often the system will interrupt the student or how often it will probe him about misconceptions. At this level, a choice is made between approaches that would diagnose the student's knowledge (*tutor*) or introduce a new topic (*introduce*).

At the second level, the pedagogy is refined into a strategy, specifying the approach to be used. The choice here might be between exploring the student's competence by questioning him, or by describing the facts of the topic without any interaction. At the lowest level, a tactic is selected to implement the strategy. For instance, if the strategy involves questioning the student, the system can choose from a half-dozen alternatives—for example, it can question the student about a specific topic, the dependency between topics, or the role of a subtopic. Again, after the student has given his answers, the system can choose from among eight ways to respond, three of which are correcting the student, elaborating on his answer, or, alternatively, barely acknowledging his answer.

The tutoring component presently contains 40 states similar to the state of an augmented transition network, or ATN. Each state is organized as a Lisp structure with slots for functions that are run when the state is evaluated. The slots define such things as the specifications of the text to be uttered, the next state to go to, or how to update the student and discourse models. The DMN is traversed by an iterative routine that stays within a predetermined space of paths from state to state.

The key point about this control structure is that its paths are not fixed; each default path can be preempted at any time by a meta-rule that moves Meno-tutor onto a new path, which is ostensibly more in keeping with student history or discourse history. The action of the meta-rule functionally corresponds to the high-level transitions observed in human tutoring. Figure 2 represents the action of two meta-rules, one at the strategic and one at the tactical level. The ubiquity of the meta-rules—the fact that virtually any transition between tutoring states may potentially be preempted—represents an important deviation from the standard control mechanism of an ATN. Formally, the behavior of Meno-tutor could be represented within the definition of an ATN; however, the need to include arcs for *every* meta-rule as part of the arc set of every state would miss the point of our design.

The system presently contains 20 meta-rules; most originate from more than one state and move the tutor to a single, new state. The preconditions of the meta-rules determine when it is time to move off the default path. They examine such data structures as the student model (for example, Does the student know a given topic?), the discourse model (for example, Have enough questions been asked on a given topic to assess whether the student knows it?), and the domain model (for example, Do related topics exist?). Two meta-rules are described in more detail in the next section and in the following informal notation (Example 9):

S1-EXPLORE—a Strategic Meta-Rule
From: teach-data
To: explore-competency
Description: Moves the tutor to begin a series of shallow questions about a variety of topics.
Activation: The present topic is complete and the tutor has little confidence in its assessment of the student's knowledge.
Behavior: Generates an expository shift from detailed examination of a single topic to a shallow examination of a variety of topics on the threshold of the student's knowledge.

T6-A.IMPLICITLY—a Tactical Meta-Rule
From: explicit-incorrect-acknowledgment
To: implicit-incorrect-acknowledgment

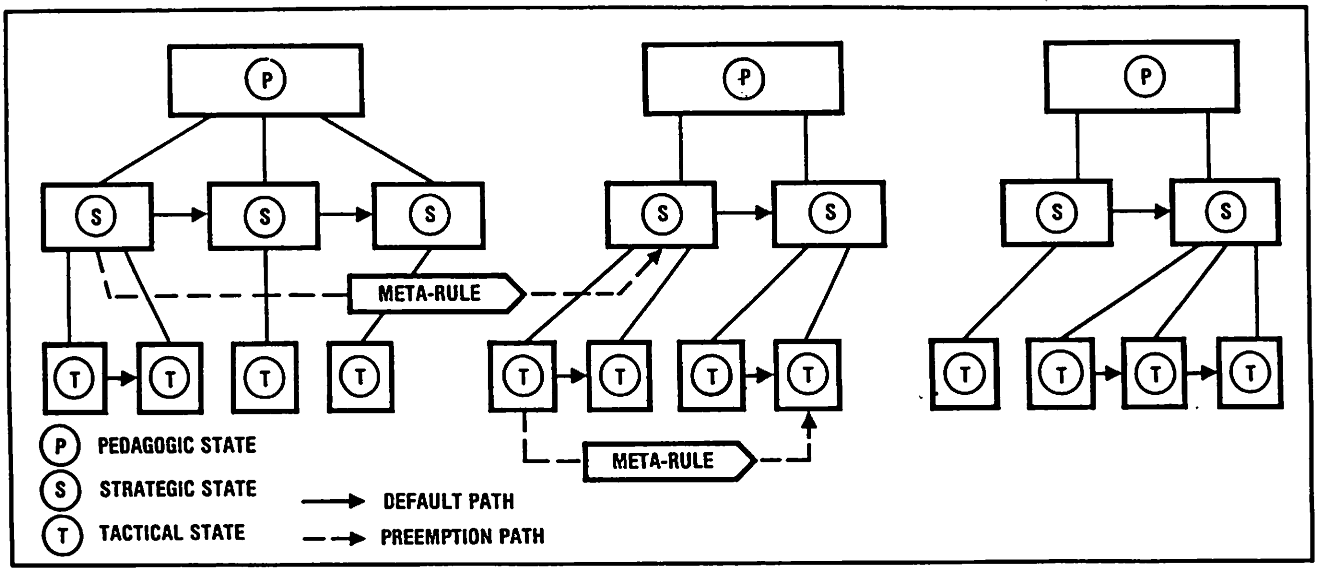

Figure 2. The action of the meta-rules.

Description: Moves the tutor to utter a brief acknowledgment of an incorrect answer.
Activation: The wrong answer threshold has been reached and the student seems confused.
Behavior: Shifts the discourse from an explicit correction of the student's answer to a response that recognizes, but does not dwell on, the incorrect answer. (9)

An example of discourse planning

Here we provide an explicit view of the way the decision units and meta-rules interact in the tutoring process. We describe the generation of a portion of the discourse in Example 1, which is reproduced here in Example 10:

Tutor:	No, it is rather mild. Can you guess about the rainfall there?
Student:	Normal, I guess.
Tutor:	Well, the Japan Current, which starts in the Southeastern Pacific, flows along the coast of Japan and across the North Pacific, ending up off the coast of Washington and Oregon. How do you think that current affects the climate there? (10)

The example discourse begins after the student's second incorrect answer. Figures 3 through 5 show six "Snapshots" of Meno-tutor's passage through a small portion of the discourse management network as it plans and generates the sample discourse.

The tutor begins in the state *explicit-incorrect-acknowledgment* (snapshot 1, Figure 3); this is a tactical state, the principal action of which is to say something, in this case "No." Having said this, the tutor still has "control" of the discourse and can continue to elaborate its response to the student's wrong answer. In the present design, there is no default path out of *explicit-incorrect-acknowledge* at the tactical level. With a different set of rules, the tutor might, for example, continue speaking, or it might reinforce the student's answer, perhaps by repeating it or elaborating a portion of it. But because our rules say that the best thing to do at this point is to move to a higher planning level and to consider reformulating either the strategy or the pedagogy of the utterance, the tutor returns to the strategic level and to the parent state, *teach-data* (indicated by the "up" arrow in snapshot 1 of Figure 3).

Once in *teach-data,* we take the default path down to the tactical level to *teach-specific-data*. In general at this point, a different meta-rule might have applied to take the tutor to a more particular tactical state, but in this case, that did not happen. At *teach-specific-data,* as in any tactical state, the tutor says something, and, in this case, it extends the utterance already begun with "No." The utterance is constructed from the specification built into this decision unit and individualized by the values its elements have in this domain and at this point in the discourse. The specification is specific-value (current-topic), where current-topic has been carried forward from the previous ply of the discourse and is still "the climate in Washington and Oregon." The attribute value of this topic is "rather mild" (a canned phrase) and Meno-tutor renders it in this discourse context (that is, "full sentence") as "It's rather mild."

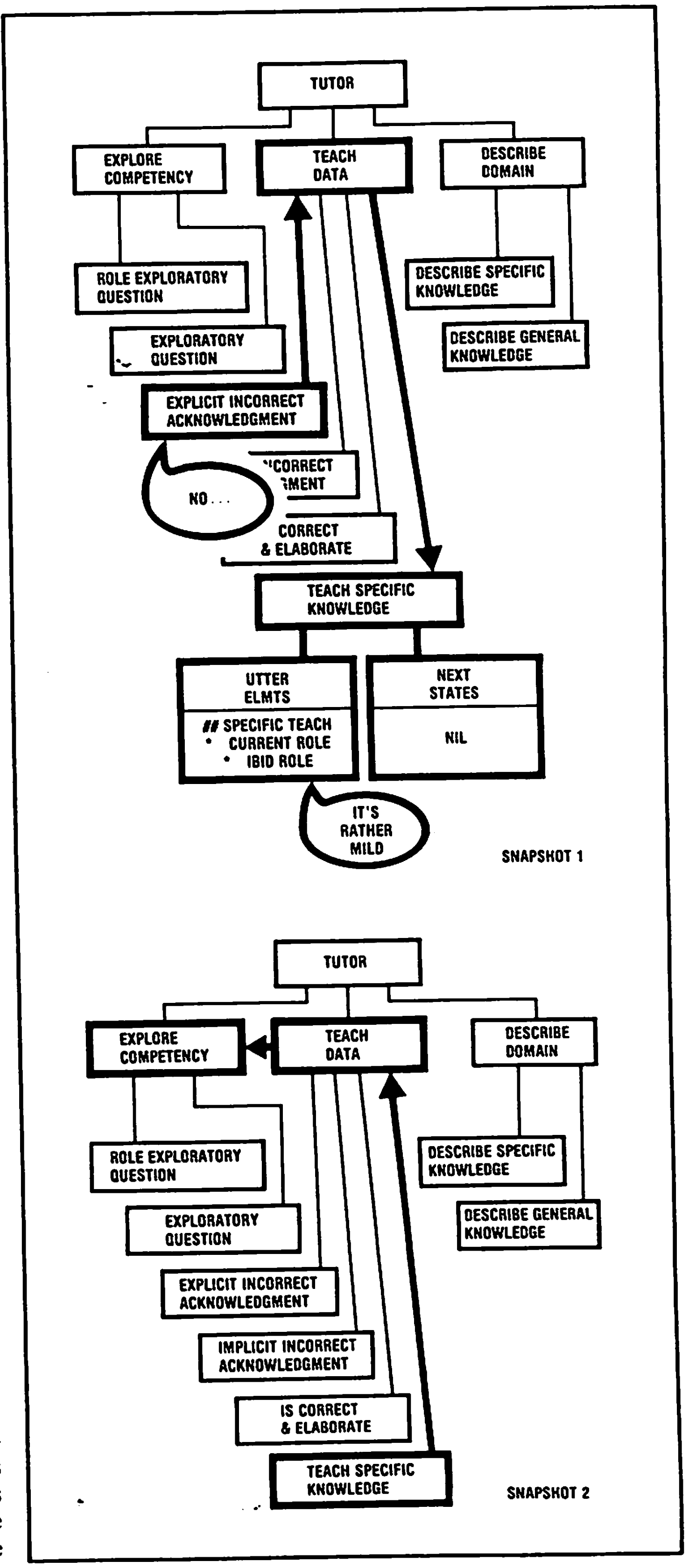

Figure 3. Movement of the tutor through the DMN during the planning and production of the utterances in Example 10. The tutor explicitly acknowledges the student's first wrong answer and then teaches about a specific topic in the knowledge base (snapshot 1). After the fault movement to the strategy *teach-data*, the system moves to explore the student's competency (snapshot 2).

From *teach-specific-knowledge* there is no default path and the tutor moves up again to *teach-data* (Figure 3, snapshot 2). This time, however, the context has changed and before *teach-data* can be evaluated, a meta-rule takes the tutor to a different decision-unit. The context has changed because the topics brought up until this point in the discourse have been answered or resolved.

In detail, what happened was that, when the tutor supplied the correct answer to its own question (i.e., "It's rather mild"), the DMN register *question_complete* was set, satisfying one of the preconditions of the meta-rule, S1-EXPLORE (see Example 9). The other precondition for this meta-rule was already satisfied, namely, that some topics related to the current topic remain to be discussed (as indicated by another register). When S1-EXPLORE is triggered, it moves the tutor to *explore-competency,* in effect establishing that previous topics are complete and that a new topic can be explored. The next most salient topic in the knowledge base is "rainfall in Washington and Oregon," and it becomes the current topic.

Once in *explore-competency,* the tutor takes a default path to the tactical level and to *exploratory-question* (Figure 4, snapshot 3). Once again, at the tactical level the tutor says something, in this case, further questioning topics on the threshold of the student's knowledge. This time the utterance is constructed from the specification built into *exploratory-question,* which has been individualized by the values at this point in the discourse. The specification is question-model (current-topic), where current-topic has been changed to "rainfall in Washington and Oregon" at the time the meta-rule was enabled, as mentioned above. The utterance put out by Meno-tutor is "Can you guess about the rainfall there?"

At this point, Meno-tutor moves to a default path and enters the tactical state *evaluate-input,* which receives and evaluates the student's answer (not shown). His answer is wrong a second time, and the default path moves the tutor, once again, to *explicit-incorrect-acknowledge,* where it would normally correct the student, as before. However, this state is not evaluated because the context is different and a new meta-rule, T6-A.IMPLICITLY (Example 9), fires, moving the tutor to a new decision-unit (Figure 4, snapshot 4). The context change is two-fold: First, the student seems confused, and second, the test for wrong-answers-threshold is met.

Recognizing a confused student is admittedly a subjective and imprecise inference for a machine tutor. In this implementation, we chose to measure the student's confusion as a function of the number of questions asked, the number of incorrect responses given, and the extent to which the student's frontier of knowledge has been explored. In the example discourse, two questions were asked, two answered incorrectly, and the student's frontier of knowledge barely explored. Therefore, the student was judged to be confused and the meta-rule T6-A.IMPLICITLY triggered, forcing the system to move to the tactical state *implicit-incorrect-acknowledgment.* This move causes the tutor to utter a refinement of its default response; instead of correcting the student, as the default response of the previous utterance did, text generated from this state implicitly recognizes, but does not dwell on, the incorrect answer and the tutor says "Well, . . ."

There is no default path from *implicit-incorrect-acknowledgment* and the tutor moves up to *teach-data* (snapshot 5, Figure 5). Once again, a meta-rule takes the tutor to a new strategic decision unit, *describe-domain.* The context in this case is that the threshold of wrong

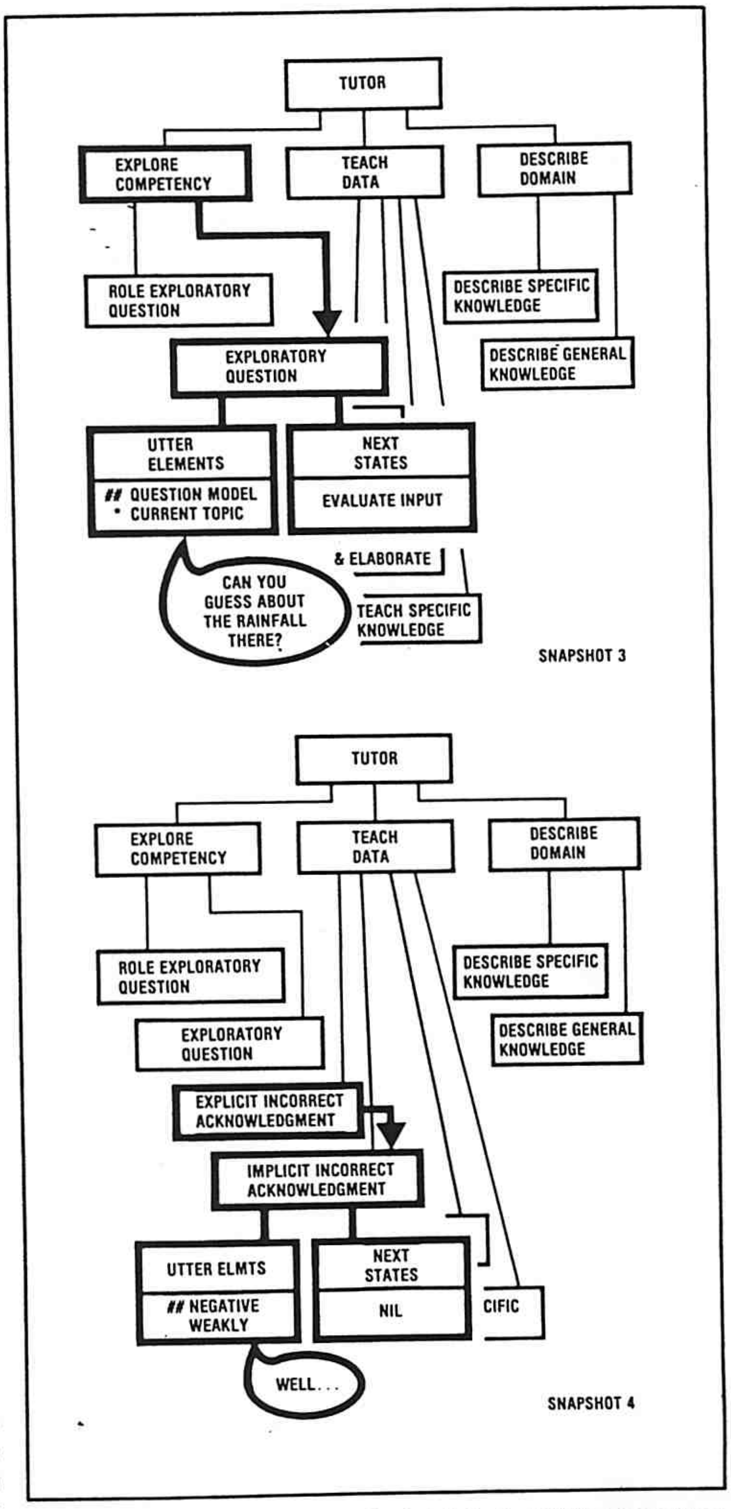

Figure 4. The tutor moves to a tactical state in the DMN and questions the student about a topic on the threshold of his knowledge (snapshot 3). After the student's second wrong answer, a meta-rule moves the system to a tactical state which utters an *implicit acknowledgment* of the incorrect answer (snapshot 4).

answers has been met (as recorded by a register) and at least one topic exists in the knowledge base ("Japan Current") that is linked to the major topic (the "climate in Washington and Oregon"). Based on the first fact, the system infers that the present strategy, *teach-data*, has been ineffective; based on the second fact, it infers that an undiscussed geographical factor remains in the knowledge base, which, if described, could enable the student to infer the answer to the original question. S3-DESCRIBE is therefore triggered, moving the tutor to *describe-domain*. The action of this meta-rule terminates the interactive question/ answer approach and begins a lengthy descriptive passage about the single topic, the "Japan Current."

From *describe-domain*, the tutor takes the default path to the tactical level to *describe-specific-knowledge* (Figure 5, snapshot 6) and prepares to speak. The utterance specification in this state is specific-describe (current-topic). As mentioned above, current-topic is now "Japan Current" and specific-describe has the effect of enunciating each attribute value of a specific topic in the knowledge base. Thus, the description realized by Meno-tutor is "the Japan Current, which starts in the Southeast Pacific, goes along the coast of Japan and across the North Pacific, ending up off the coast of Washington and Oregon."

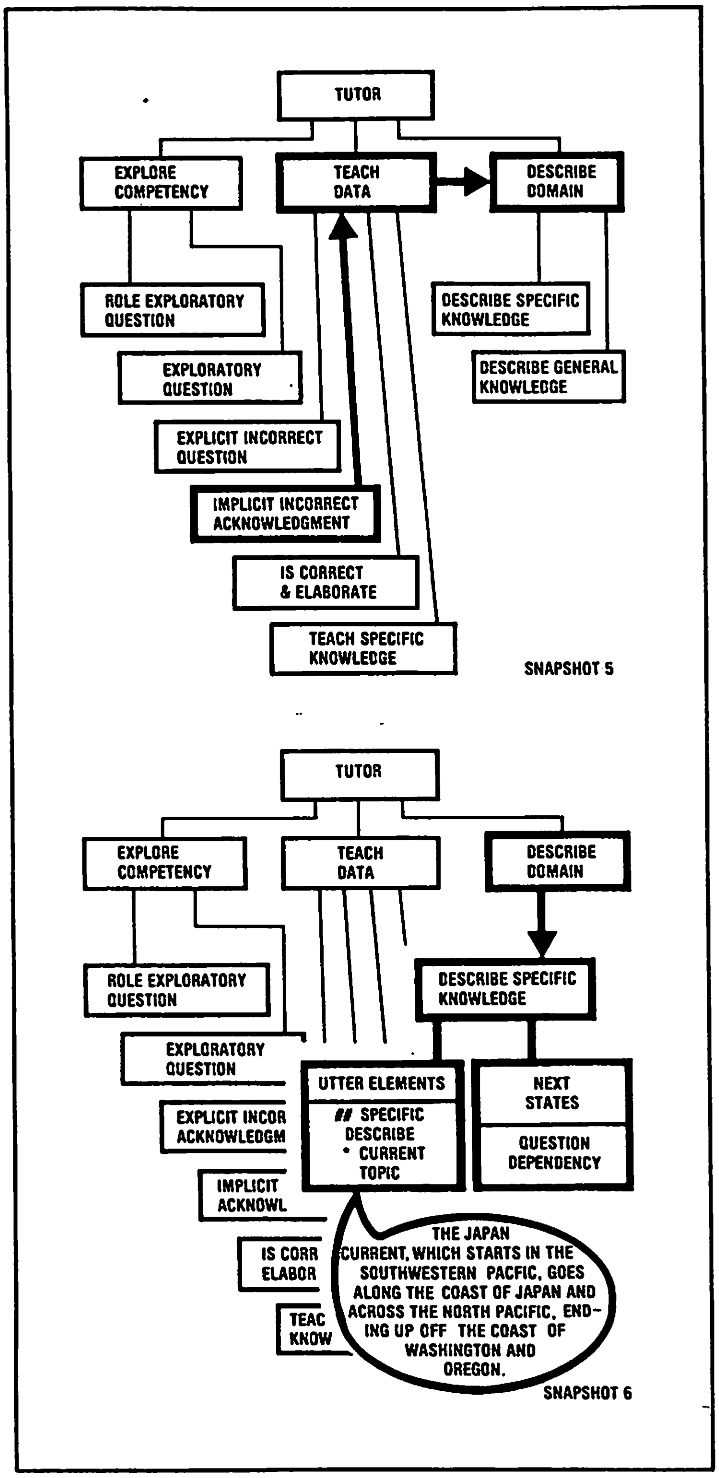
TUTOR
EXPLORE COMPETENCY
TEACH DATA
DESCRIBE DOMAIN
ROLE EXPLORATORY QUESTION
DESCRIBE SPECIFIC KNOWLEDGE
EXPLORATORY QUESTION
DESCRIBE GENERAL KNOWLEDGE
EXPLICIT INCORRECT QUESTION
IMPLICIT INCORRECT ACKNOWLEDGMENT
IS CORRECT & ELABORATE
TEACH SPECIFIC KNOWLEDGE
SNAPSHOT 5
TUTOR
EXPLORE COMPETENCY
TEACH DATA
DESCRIBE DOMAIN
ROLE EXPLORATORY QUESTION
DESCRIBE SPECIFIC KNOWLEDGE
EXPLORATORY QUESTION
UTTER ELEMENTS
// SPECIFIC DESCRIBE • CURRENT TOPIC
NEXT STATES
QUESTION DEPENDENCY
EXPLICIT INCOR ACKNOWLEDGM
IMPLICIT ACKNOWL
IS CORR ELABOR
TEAC KNOW
THE JAPAN CURRENT, WHICH STARTS IN THE SOUTHWESTERN PACFIC, GOES ALONG THE COAST OF JAPAN AND ACROSS THE NORTH PACIFIC, ENDING UP OFF THE COAST OF WASHINGTON AND OREGON.
SNAPSHOT 6

Figure 5. Default movement back to the strategy of *teach-data* is interrupted by a meta-rule, which moves the system to a new strategy, that of describing topics in the knowledge bases (snapshot 5). The system moves, by default, to the tactic of describing a specific topic related to earlier discourse topics (snapshot 6).

We have suggested that *because* tutoring can be affected by problems in communication, customization of a system's response to the individual student is central to its effectiveness. We have also described how the author of a tutoring system might begin to adapt a system's response to the student's level of knowledge. The data and control structures of Meno-tutor were described to show how a tutoring system can use AI techniques to model the student, a domain, and the teaching strategies in the planning and generation of its discourse. ✻

References

1. A. Stevens, A. Collins, and S. Goldin, "Diagnosing Student's Misconceptions in Causal Models," *Int'l J. Man-Machines Studies*, Vol. 11, Jan. 1979; also in *Intelligent Tutoring Systems*, Sleeman and Brown, eds., Academic Press, Cambridge, Mass., 1982.

2. J. Allen, "A Plan-Based Approach to Speech Act Recognition," *Computational Models of Discourse*, M. Brady and R. Berwick, eds., MIT Press, Cambridge, Mass., 1983.

3. J. S. Brown, R. Burton, and A. Bell, "SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An example of A.I. in C.A.I.)," *Int'l J. Man-Machine Studies*, Vol. 7, 1977.

4. W. Clancey, "Tutoring Rules for Guiding a Case Method Dialogue," *Int'l J. Man-Machine Studies*, Vol. 11, 1978; also in *Intelligent Tutoring Systems*, Sleeman and Brown, eds., Academic Press, Cambridge, Mass., 1982.

5. R. Burton and J. S. Brown, "An Investigation of Computer Coaching for Informal Learning Activities," *Int'l J. Man-Machine Studies,* Vol. 11, 1978; also in *Intelligent Tutoring Systems,* Sleeman and Brown, eds., Mass., 1982.

6. J. S. Brown and R. Burton, "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills," *Cognitive Sciences,* Vol. 2, 1978, pp. 155-192.

7. B. Woolf, "Context-Dependent Planning in a Machine Tutor," PhD dissertation, Dept. of Computer and Information Science, University of Massachusetts, Amherst, Mass., 1984.

8. B. Woolf and D. McDonald, "Human-Computer Discourse in the Design of a Pascal Tutor," *CHI 83: Human Factors in Computer Systems,* ACM, New York, 1983.

9. J. Bonar, "Understanding the Bugs of Novice Programming," PhD dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, Mass., 1984.

10. E. Soloway, B. Woolf, P. Barth, and E. Rubin, "MENO-II: An Intelligent Tutoring System for Novice Programmers," *Seventh Int'l Joint Conf. Artificial Intelligence,* Vancouver, Canada, 1981.

11. D. McDonald, "Natural Language Generation as a Computational Problem: An Introduction," *Computational Models of Discourse,* M. Brady and R. Berwick, eds., MIT Press, Cambridge, Mass., 1983.

Beverly Woolf is an assistant professor in the Department of Computer and Information Sciences, University of Massachusetts. Her research interests include artificial intelligence, cognitive science, knowledge representation, and human-computer interfaces for explanation and teaching.

She received her BA in physics from Smith College and her MS and PhD in computer science from the University of Massachusetts.

David D. McDonald is on the faculty of the Department of Computer and Information Science at the University of Massachusetts. His primary research area is natural language processing, and he also does research on knowledge representation, planning, and high-performance programming environments; his work on language generation is well known. He and his group have active projects in machine tutoring, description and explanation, presentations by intelligent interfaces, and data-directed control and planning.

After studying linguistics as an undergraduate at MIT, he joined the MIT Artificial Intelligence Laboratory, where he stayed until receiving his PhD in 1980.

Questions about this article can be directed to Beverly Woolf at the Department of Computer and Information Sciences, University of Massachusetts, Amherst, MA 01003.

APPLIED AI SYSTEMS DESIGN PROFESSIONALS

The Advanced Information Technology Applications Center at SRI INTERNATIONAL, on the San Francisco peninsula, is looking for professionals with backgrounds in Artificial Intelligence to work on the design and implementation of prototype applied AI systems in the context of user specific problem domains.

Computer Scientist

Candidate will have familiarity with knowledge-based systems and natural language processing. Proficiency in LISP and PROLOG required. 1-10 yrs. experience in the above along with a BSEE or BSCS; MSCS preferred.

Senior Computer Scientist

In addition to the credentials for a Computer Scientist, 10 or more years relevant work experience is required.

Program Manager

Plan, structure, staff and oversee the applied AI activity in the Advanced Information Technology Applications Center. Experience in the design and implementation of knowledge based systems and familiarity with other AI research areas such as image understanding, natural languages and voice recognition essential. Must be able to implement in LISP and PROLOG. MS/EE or CS; PhD preferred.

SRI INTERNATIONAL is a non-profit research, development and consulting organization. For immediate consideration, please send resume to Michael Patrick, Dept. COM-9/84, Room EL-102, SRI INTERNATIONAL, 333 Ravenswood Avenue, Menlo Park, CA 94025. U.S. Citizenship is required. An equal opportunity employer.

SRI International®