Modeling Human Teaching Tactics in a Computer Tutor*

Mark G. Core, Johanna D. Moore, Claus Zinn, and Peter Wiemer-Hastings

HCRC, University of Edinburgh
Edinburgh EH8 9LW, UK
markc,jmoore,zinn,peterwh@cogsci.ed.ac.uk

Abstract. Previous psychological research has shown that students must construct knowledge themselves to learn most effectively. This means tutors should not simply give explanations or tell the student the correct answer to a question. Tutors and students should co-construct explanations and tutors should walk students through lines of reasoning. These activities unfold over multiple turns and tutors must be flexible enough to deal with (i) failure (students may answer a tutor question wrong or the whole tactic may not be working), (ii) interruptions (students may interrupt with a question) and (iii) the need to revise their tactics (student behavior may indicate that the tutor can skip steps in an explanation). We discuss these problems in relation to tutorial dialogue planning and propose tentative solutions.

1 Motivation

Research on student learning has shown that students must construct knowledge themselves to learn most effectively [4, 5]. Studies also show that one-on-one human tutoring is more effective than other modes of instruction. Tutoring raises students' performance as measured by pre and post-tests by 0.40 standard deviations with peer tutors [6] and by 2.0 standard deviations with experienced tutors [1]. What is it about human tutoring that facilitates this learning? Many researchers argue that it is the collaborative dialogue between student and tutor that promotes the learning [15, 9, 12]. Through collaborative dialogue, tutors can intervene to ensure that errors are detected and repaired and that students can work around impasses [16]. The consensus from these studies is that experienced human tutors maintain a delicate balance, allowing students to do as much of the work as possible and to maintain a feeling of control, while providing students with enough guidance to keep them from becoming too frustrated or confused.

For an intelligent tutoring system (ITS) to imitate these successful tutors, it must support:

1. unconstrained natural language input — other modes of input (menus, fill-in-the-blank forms) change the task from knowledge construction to correct answer recognition.

* The research presented in this paper was supported by Grant #N00 014-91-J-1694 from the Office of Naval Research, Cognitive and Neural Sciences Division.

2. "extended" tutoring strategies (i.e., strategies that unfold over multiple dialogue turns) — allowing tutors and students to co-construct explanations and allowing tutors to lead students through a line of reasoning point by point.

To support unconstrained natural language and multi-turn teaching tactics, a computer tutor must be able to deal with:

1. failure — (i) the tutor may not understand a student response; (ii) the student may answer a tutor question in an unexpected manner; and (iii) the tutor's teaching tactic may not be working.
2. interruption — the student may interrupt with a question.
3. the need to revise tactics — a student may skip steps in an explanation.
4. the need to disambiguate student meaning.

To test our ideas about building such tutors, we have been investigating tutoring basic electricity and electronics (BE&E). Our starting point is a course on basic electricity and electronics developed with the VIVIDS authoring tool [17]. Students read textbook-style lessons written in HTML and then perform labs using the graphical user interface (GUI) shown in Fig. 1. In [19], Rosé *et al.* describe an experiment where students go through these lessons and labs with the guidance of a human tutor. The video signal from the student's computer was split so the tutor who was hidden behind a partition could watch the student's progress. The tutor and student were allowed to type messages to each other through a chat interface. We will refer to the logs of this chat interface as the BE&E dialogues. We use the BE&E dialogues to identify teaching tactics to be used by our tutor and plan to use them to train our system.

In this paper, we focus on how such teaching tactics are best implemented in a computer tutor's dialogue planner. In Sect. 2, we discuss previous work in dialogue planning. We identify several requirements a dialogue planner must meet and tentatively choose a two-level approach with recursive transition networks handling communication management and a hierarchical reactive planner performing content planning using domain independent teaching tactics. In Sect. 3, we show an example demonstrating why reactive planning is necessary. Section 4 contains general discussion.

2 Previous Work

Our goal is to support coherent tutorial dialogue that effectively promotes student learning. There are two aspects to this goal; the tutor must plan how to teach a student a particular concept (content planning) as well as maintain the conversation (communication management planning). Communication management refers to signaling topic shifts, acknowledging and accepting student utterances, and handling cases where one of the conversants did not fully hear or understand the other. One should keep the two types of planning separate. Communication management operators can be used to provide a level of abstraction.

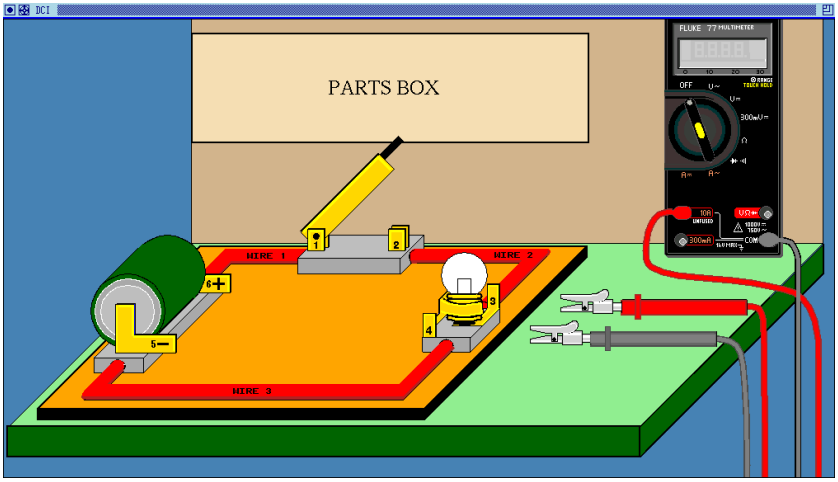

Fig. 1. BE&E Graphical User Interface

Content plan operators can then be written to capture teaching tactics without having to include details such as acknowledgment and acceptance. The CIRC-SIM tutor [10] does not make this separation but the EDGE explanation system [3] has separate content operators and communication management operators (called discourse operators in [3]).

We have tentatively chosen to go one step further and have a separate communication management planner and content planner following the architecture of the Autoroute spoken dialogue system [14] and the three-level architecture commonly used for planning in robotics [2]. It is not clear if the missing third level, physical control of the robot, has an analogy in tutorial dialogue planning. In Autoroute, communication management is considered low-level and is performed by a series of recursive transition networks. Each network implements a conversational game such as asking a question or giving information. Each game handles the communication management associated with the system's utterances. Such a technique has the potential to be faster than having one traditional planner deal with both content and communication management actions. In addition, as described in [18], Bayes nets can be used to navigate the conversational games. One can train the Bayes nets on a corpus to determine when, for example, to explicitly acknowledge. This is not a simple decision and should be based on confidence scores from the speech recognizer and natural language parser and judgment of the input (in the case of Autoroute, judgment is made with respect to whether the user utterance is a valid database command).

The Autoroute content planner is high-level, and treats conversational games as atomic actions having preconditions and postconditions. It uses the situation calculus to combine these games in a STRIPS-like manner into a plan to achieve its major goal (constructing a query to a route planning system).

To plan an entire dialogue in this manner, one must assume that the other conversant will respond as expected. In tutorial dialogues especially, this assump-

tion is likely to be wrong; a student may not answer a question correctly and may not be easily led "back on track". Or a student may give more information than was requested and part of the tutor's plan may be unnecessary. Thus, we follow [11] and use a reactive planner to implement content planning in our tutor. As in the EDGE explanation system [3], we will plan in a depth-first manner until an executable action is reached; we will not elaborate future parts of the plan until necessary.

To help build the library for this planner, we will identify and annotate the teaching tactics in the BE&E dialogues. Since we have pre- and post-test scores for the students in our corpus, we can evaluate the effectiveness of the teaching tactics used. This annotation will help us gain a more detailed understanding of the capabilities needed of the dialogue planner, and test our decision to use a hierarchical reactive planner for content planning and recursive transition networks for communication management. One might ask whether reactive planning is necessary at all, and whether Autoroute's original STRIPS-style planner would be adequate for dialogue planning. In Sect. 3, we go through an example showing the need for a reactive planner.

Our teaching tactics are specific to procedural tasks but our goal is make them otherwise domain independent. The CIRCSIM tutor [10] and EDGE explanation system [3] have a mixture of domain dependent and domain independent teaching tactics in their dialogue plan libraries. To test whether we have been successful in developing a domain independent plan library, we will try applying it to another procedural tutoring domain such as guiding students through chemistry experiments or medical training.

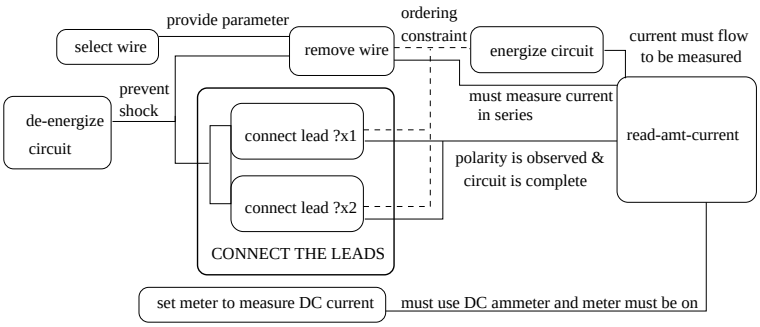

Fig. 2. Domain Plan for Measuring Current

3 Proposed Dialogue Planner

Appendix A depicts the start of a BE&E dialogue. Utterances labeled GUI are text displayed as part of the graphical user interface while utterances labeled Student and Tutor were produced by the human student and tutor. In this dialogue, the student is supposed to be measuring current; the plan for measuring current is shown in Fig. 2. Preconditions are depicted as lines connecting actions.

This dialogue contains three instances where the student does not respond as expected to tutor questions (utterances 21, 28, 30-31). In utterances 28 and 30-31, the student is clearly not able to produce the right answer. Utterance 21 has three problems: it uses wrong terminology (*wire* instead of *lead*), is vague (which lead is picked?), and does not fully answer the question (there are other steps in connecting leads). In this section, we show how a two-level dialogue planning architecture can be used to deal with these unexpected responses.

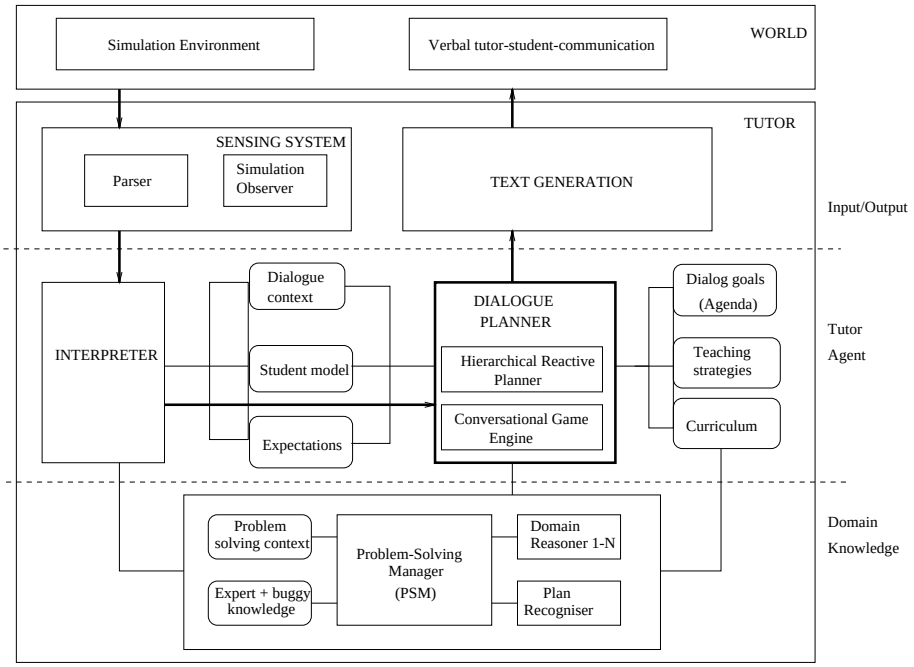

Fig. 3. The BEE Tutor Architecture

To implement the teaching tactics used by the tutor in this dialogue, we propose the architecture presented in Fig. 3. The architecture is inspired by the TRIPS dialogue system [8]. Control flows through this architecture as follows:

1. If the student types input then the parser produces an underspecified logical form that the interpreter attempts to fully specify.
2. The interpreter uses the problem solving manager, dialogue context, expectations, and curriculum to evaluate input (note, input to the tutor may simply be that the student is idle).
3. The interpreter updates the student model.
4. The interpreter may send messages directly to the dialogue planner (*e.g.*, an evaluation of the student's answer to a question or an alert when one of the values in the student model falls below threshold)

5. If a conversational game is in progress, then the conversational game engine runs it, else the reactive planner is run to load a new conversational game.
6. The reactive planner is run if a conversational game ends or there is unexpected student input (*e.g.*, the student says *red wire* instead of *red lead*).

The problem solving manager matches student actions to correct and incorrect (buggy) plans. Plans are incorrect with respect to not achieving a particular goal or achieving a goal inefficiently. The student model is updated based on student actions as well as student inaction; belief that the student knows the next step of the current plan decreases with the time it is taking him[1] to perform this action. We are assuming a probabilistic student model such as the one presented in [7] where domain actions and concepts are connected in a Bayes net. An example connection would be that knowing how to attach the leads suggests knowing that a circuit must be complete for current to flow.

In the following paragraphs, we explain how our tutor could simulate the human tutor in the dialogue of appendix A. In general the computer tutor will have many decisions to make: should a student just be told a piece of information, asked to provide this information, or shown this information through a multiturn teaching tactic. Here, we make the same decisions as the human tutor.

Our dialogue planner uses a data structure called an agenda to store goals. We are investigating whether the agenda should be a stack or whether the planner needs to modify and delete arbitrary goals from the agenda. One reason for the planner to edit the agenda is to switch between goals. If the student is not making progress on one goal then the tutor may want to address another goal for a while.

The tutor's curriculum contains a list of tutoring goals. When the tutor starts up, the first unachieved tutoring goal for a particular student is placed on the dialogue planner's agenda. In this case, the goal is `(student-performs (measure current))`.[2] The dialogue planner constructs a plan for achieving this goal using operators from its domain independent plan library (the operators relevant to utterances 1-19 are shown in Fig. 4). To simulate the high-level structure of the dialogue in appendix A, we use the TEACH-STEP-BY-STEP operator. This operator has both constraints and preconditions. Unlike preconditions, constraints cannot be made true through the application of other operators. The preconditions state that to get the student to perform an action (measure current), the tutor must get the student to perform all the substeps of this action. Note this decomposition is provided by the problem solving manager. None of the operators used by the dialogue planner are domain dependent. The actions (conversational games) associated with TEACH-STEP-BY-STEP (in this case, none) are pushed onto the agenda followed by the preconditions of the operator (A-G in the agenda below). For presentation purposes, subgoals in the agenda are listed under the goals they support. The arrow indicates the next goal to be addressed.

[1] We refer to the student as he and the human tutor as she.

[2] The curriculum can contain arbitrarily complex goals encoding not only concepts to be taught but a preferred method of teaching of them.

```
TEACH-STEP-BY-STEP ?a

effects: (student-performs ?a)
constraints:  (AND (step ?a) (not (primitive ?a)))
preconditions: (foreach (?substep (PSM-ASK DECOMPOSITION ?a))
                  (student-performs ?substep))

TEACH-NEXT-STEP ?a

effects: (student-performs ?a)
constraints:  (AND (step ?a) (not (primitive ?a)))
preconditions: (AND (student-knows (next ?a))
                    (student-knows (how-to-perform ?a)))

PRIME-NAME-NEXT ?a

effects: (primed (next ?a))
constraints: (AND (set instruction-list (PSM-ASK INSTRUCTIONS))
                  (set actions-performed-list (PSM-ASK ACTIONS))
                  (step ?a))
preconditions: (AND (salient (instructions instruction-list))
                    (salient (actions-performed actions-performed-list)))

ASK ?a

effects:  (student-knows ?a) (student-states ?a) (salient ?a)
precondition: (primed ?a)
action: (ASK-GAME ?a)

INSTRUCT ?a

effects: (student-performs ?a)
action:  (INSTRUCT-GAME ?a)

INSTRUCT2 ?a

effects: (student-performs ?a) (student-performs ?b)
action:  (INSTRUCT-GAME ?a ?b)
```

Fig. 4. Dialogue Planning Operators

```
AGENDA:  (student-performs (measure current))
         -> A. (student-performs (de-energize circuit))
            B. (student-performs (set-meter-to-dc-current))
            C. (student-performs (select wire))
            D. (student-performs (remove wire))
            E. (student-performs (connect leads))
            F. (student-performs (energize circuit))
            G. (student-performs (read-amt-current))
```

To address goal A, the dialogue planner chooses the INSTRUCT operator which has no preconditions and one action, running the INSTRUCT-GAME (see Fig. 5, adapted from [14]). The game is a recursive transition network where the links represent complex actions such as giving instructions or actions to the GUI. The game is started at state T0 (tutor's first turn). The conversational game engine performs the only action possible, giving instructions (utterances 1-2) using the text generator. The text generator produces natural language text from logical forms. For example, `(de-energize circuit)` would be realized as *Set the switch (i.e. the circuit switch) to off*. After producing the instructions, the game engine waits for the student's response. Interpreting the student's response is tricky because the student may do nothing (eventually the game engine must

concede failure) or the student may perform more than one action (the game engine must decide when the student is done with his response). This example is relatively simple as the student performs the correct response causing the game engine to move to state T1. The game engine then decides to make an acceptance (utterance 3) ending the game. Since the goal of getting the student to de-energize the circuit was accomplished, the reactive planner pops goal A from the agenda. Goals B and C are addressed successfully in the same manner.

The dialogue planner addresses goals D and E using the INSTRUCT2 operator.[3] The INSTRUCT-GAME is run producing utterance 8. After a certain length of time, the conversational game engine must decide that the student is not going to connect the leads. The student's actions are classified as incorrect; an explicit acknowledgment is made in utterance 9 (*OK*) and the game ends in failure. The game was used to address goals D and E. D was achieved while E was not. The reactive planner pops goal D from the agenda.

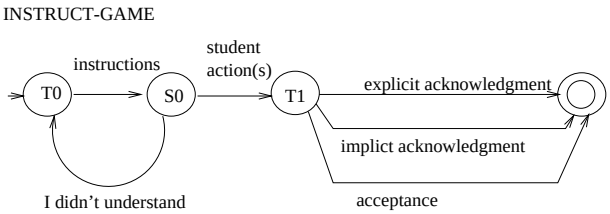

Fig. 5. INSTRUCT-GAME

We simulate the teaching tactics displayed in utterances 9-20 by first applying the TEACH-NEXT-STEP operator which says the student must know what step is next (goal E1, in the agenda shown in Fig. 6) and how to perform it (goal E2) in order to execute the action. A precondition of asking the student to identify the next step (goal E1.2) is priming the student through PRIME-NAME-NEXT (goal E1.1). PRIME-NAME-NEXT involves making the instructions salient (goal E1.1.1) and making the student's actions salient (goal E1.1.2). The idea is that the student will then be primed to answer the question: *what is the next step in the plan?*. Note in applying the ASK operator to address goal E1.1.1, we assume we do not need to prime the student in order to ask what the instructions were. This assumption can later be retracted if it turns out to be false. This is a fairly simple example of a directed line of reasoning where a student is lead to a conclusion through a series of reasoning steps [13]. Also note, none of the operators used are domain dependent. The problem solving manager provides a list of the actions the student performed as well as the instructions given by the tutor.

[3] In this preliminary investigation we use a separate operator, INSTRUCT2, to address two goals at once. In future work, we plan to develop a more general INSTRUCT operator that allows more than one goal to be addressed.

Some goals on the agenda are tied to the student model as suggested by [3]. So if the model indicates that the student knows he must connect the leads, then the tutor will not bother hinting and asking about what must be done next. Some preconditions of the operators in Fig. 4 involve making certain information salient. For example, even if the tutor thinks the student knows the instructions, the tutor will re-iterate them ensuring a coherent discussion.

```
E) (student-performs (connect leads))
E1) (student-knows (next (connect leads)))
   E1.1) (primed (next (connect leads)))
   E1.1.1) (salient (instructions instruction-list))
-> E1.1.1.1) (ASK-GAME (instructions instruction-list))
   E1.1.2) (salient (actions-performed actions-performed-list))
   E1.2) (ASK-GAME (next step-in-plan))
E2) (student-knows (how-to-perform (connect leads)))
F. (student-performs (energize circuit))
G. (student-performs (read-amt-current))
```

Fig. 6. Agenda Just before Utterance 10 Is Produced

To execute the ASK-GAME associated with E1.1.1.1 we start by asking the question seen in utterances 10-11. The ASK-GAME is the same as the INSTRUCT-GAME except that a question is asked instead of instructions given and student actions consist of responses. The student makes the reply seen in utterance 12 and the tutor utters an explicit acknowledgment, *okay*, completing the game. E1.1.1.1 is popped off the agenda, and since the student answered correctly, E1.1.1 is popped off the agenda as well. ASK-GAMEs are played for utterances 14-16 and 17-19 and E1.1.2, E1.1, E1.2, and E1 are popped off the agenda: the student knows he must connect the leads.

To address goal E2, the tutor uses the ASK operator and asks *And how are you going to do that?*. There are three problems with the student's answer, *pick one of the wires on the right of the picture*: (1) the interpreter with the help of the problem solving manager determines that *one of the wires on the right of the picture* is vague and can refer to either the black lead or the red lead; (2) the problem solving manager (after consulting the curriculum) knows that the student should use the term *lead* instead of *wires on the right of the picture*; and (3) the problem solving manager identifies the answer as incomplete (it does not say where to attach the first lead or anything about the other lead). The interpreter encodes these problems as dialogue planning goals: (1) student states which lead to attach, (2) student learns the term lead, (3) student states the remaining steps involved in connecting the leads. Goal 2 is tangential; we later see the tutor ignores the incorrect use of *wire* for *lead* in utterances 25, 28, 31, and 40.

Due to space constraints, we can only give high-level details about the rest of the dialogue. Goal 2 is addressed indirectly by utterance 22, *You mean the leads of the multimeter?* Goal 3 is split into two parts: (a) specifying the missing parameter, the attachment point of the first lead and (b) describing the second step of connecting the leads, connecting the second lead. Utterance 24 addresses goal 1 and part (a) of goal 3. The goal behind utterances 27-32 is to bring the

reading-amount-of-current step (the last step of the requested action) into focus and then to ask about unsatisfied preconditions of this action (utterances 35 and 37) resulting in the student describing the missing action in the plan. Utterances 27-32 are notable because the tutor must switch tactics at utterance 32 because the previous teaching tactic was not working. Notice the techniques used here apply to any complex action to be performed by a student not just connecting leads.

4 Discussion

Our overall goal is to build a computer tutor that supports constructive learning; our focus here is (1) building a dialogue planner that is quick and flexible enough to support dialogue that unfolds over multiple turns, and (2) developing a library for this planner including the human teaching tactics in our corpus. The two level architecture of the Autoroute dialogue system [14] is promising because the lower level abstracts communication management details from the higher level, content planning.

Using a reactive planner for the second level of our dialogue planner we have analyzed how this planner would handle the sample dialogue shown in appendix A. The sample dialogue contains examples where the student does not give the expected response (utterances 21, 28, and 30-31). We discussed how the tutor successfully dealt with utterance 21 in some detail. In utterance 28, the interpreter again has to generate a dialogue planner goal to try and get the student back on track. In this case, since the student has rephrased the question instead of answering it, an obvious solution is to repeat the question. When the student rephrases the question again (utterances 30-31), we must be careful not to repeat this strategy. The dialogue planner decides to try a new teaching tactic in utterance 32 and succeeds in getting the student to identify the reason for his actions.

In other dialogues, we have observed that the human tutor is careful not to perform steps of a teaching tactic that are unnecessary. When the dialogue begins, the tutor may ask the student to define an electrical source and load, and then ask them if the leads span a source or load. Later in the dialogue, the tutor generally only asks if the leads span a source or load and does not ask for any definitions.

As we continue to annotate our corpus of human-human dialogues, we will likely uncover more difficult examples requiring the dialogue planner to modify its teaching tactic as it executes it. Consider the following case: the tutor's teaching tactic is to (i) make the instructions salient, (ii) make the student's actions salient, and (iii) ask the student what steps remain. The tutor addresses goal i by asking *what are the instructions?*. The student replies *remove the wire and connect the leads. I still need to connect the leads.* At this point, the tutor should not perform steps ii and iii. By allowing the dialogue planner's agenda to be arbitrarily examined and modified, our architecture allows such examples to be handled. These types of examples are inevitable if the tutor allows un-

constrained natural language responses to questions in its multi-turn discourse plans; at any point, a student can jump ahead or request clarification. However, unconstrained natural language input and multi-turn teaching strategies are necessary for the most effective type of teaching, having students construct their own knowledge.

References

1. B. S. Bloom. The 2 Sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. In *Educational Researcher*, volume 13, pages 4–16, 1984.
2. R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental Theoretical Artificial Intelligence*, 9:237–256, 1997.
3. A. Cawsey. Explanatory dialogues. *Interacting with Computers*, 1(1):69–92, 1989.
4. M. T. H. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13(2):145–182, 1989.
5. M. T. H. Chi, N. De Leeuw, M. H. Chiu, and C. Lavancher. Eliciting self-explanations improves understanding. *Cognitive Science*, 18(3):439–477, 1994.
6. P. A. Cohen, J. A. Kulik, and C. C. Kulik. Educational outcomes of tutoring: A meta-analysis of findings. *American Educational Research Journal*, 19:237–248, 1982.
7. C. Conati, A. Gertner, K. VanLehn, and M. Druzdzel. On-line student modeling for coaching problem solving using bayesian networks. In A. Jameson, C. Paris, and C. Tasso, editors, *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 231–242. Springer Wien, 1997.
8. G. Ferguson and J. F. Allen. TRIPS: An intelligent integrated problem-solving assistant. In *Proc. of the National Conference on Artificial Intelligence (AAAI-98)*, pages 26–30, Madison, WI, July 1998.
9. B. A. Fox. *The Human Tutorial Dialogue Project: Issues in the design of instructional systems*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.
10. R. Freedman. *Interaction of Discourse Planning, Instructional Planning and Dialogue Management in an Interactive Tutoring System*. PhD thesis, Northwestern University, 1996.
11. R. Freedman. Using a reactive planner as the basis for a dialogue agent. In *Proceedings of the Thirteenth Florida Artificial Intelligence Symposium (FLAIRS '00)*, Orlando, 2000.
12. A. C. Graesser, N. K. Person, and J. P. Magliano. Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology*, 9:495–522, 1995.
13. G. Hume, J. Michael, A. Rovick, and M. Evens. Hinting as a tactic in one-on-one tutoring. *The Journal of the Learning Sciences*, 5(1):23–47, 1996.
14. I. Lewin. Autoroute dialogue demonstrator. Technical Report CRC-073, SRI Cambridge, 1998.
15. D. C. Merrill, B. J. Reiser, and S. Landes. Human tutoring: Pedagogical strategies and learning outcomes. Paper presented at the annual meeting of the American Educational Research Association, 1992.
16. D. C. Merrill, B. J. Reiser, M. Ranney, and J. G. Trafton. Effective tutoring techniques: Comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences*, 2(3):277–305, 1992.

17. A. Munro. Authoring interactive graphical models. In T. de Jong, D. M. Towne, and H. Spada, editors, *The Use of Computer Models for Explication, Analysis and Experimental Learning*. Springer Verlag, 1994.
18. S. Pulman. Conversational games, belief revision and bayesian networks. Technical Report CRC-071, SRI Cambridge, 1996.
19. C. P. Rosé, J. D. Moore, K. VanLehn, and D. Allbritton. A comparative evaluation of socratic versus didactic tutoring. Submitted for publication.

Appendix A: A BE&E Dialogue on Measuring Current

```
GUI:      1. In this exercise, you will get practice measuring current with
              a multimeter. This lab will lead you through the steps.
              Remember that when measuring current, you must power down the
              equipment first and then connect the meter in series with the circuit.
GUI:      2. Set the switch (i.e. the circuit switch) to off.
Student:  <turns off switch>
GUI:      3. Correct.
GUI:      4. Set the Multimeter to adc.
Student:  <sets the meter correctly>
GUI:      5. Correct.
GUI:      6. Choose the wire spanning the first position in the circuit where
            you want to measure current.
Student:  <selects wire 2>
GUI:      7. Correct.
GUI:      8. Remove the appropriate wire and connect the leads of the
              multimeter to the circuit.
Student:  <only removes wire 2 from the circuit>
Tutor:    9. OK,
         10. let's break this down into steps.
         11. What did the instructions ask you to do?
Student: 12. remove the appropriate wire and connect the leads
Tutor:   13. OK,
         14. and how much of this did you do?
Student: 15. i only removed the wire
Tutor:   16. Right.
Tutor:   17. So what do you still need to do?
Student: 18. connect the leads
Tutor:   19. Right.
         20. And how are you going to do that?
Student: 21. pick one of the wires on the right of the picture
Tutor:   22. You mean the leads of the multimter?
Student: 23. yes
Tutor:   24. Which one will you pick and where will you attach it?
Student: 25. i believe that I'll pick the red wire and
             attach to where I removed the beginning of the wire
Tutor:   26. OK,
         27. do you understand why you are doing that?
Student: 28. because you need to connect the red wire to the beginning of the wire
Tutor:   29. Why?
Student: 30. i'm not exactly sure.
         31. i just thought that you connect the red wire to the beginning
Tutor:   32. Do you understand what the goal of this lab is?
Student: 33. to be able to connect the multimeter to the light
             and read the number of amps are flowing
Tutor:   34. OK,
         35. in order for any amps to be flowing, what has to be true about the circuit?
Student: 36. it has to be  complete
Tutor:   37. If you do only what you said you were going to do,
             will the circuit be complete?
         38. (with the exception f the switch being open)?
Student: 39. no.
         40. i will also have to connect the red wire to the number 3 on
              the picture(above the lightbulb) i meant black wire not red
Tutor:   41. Good.  You are absolutely correct.
         42. You need to do both of those things before you press continue.
[...]
```