Supporting Constructive Learning with a Feedback Planner*

Mark G. Core, Johanna D. Moore, and Claus Zinn
HCRC, University of Edinburgh
Edinburgh EH8 9LW, UK
markc,jmoore,zinn@cogsci.ed.ac.uk

Abstract

A promising approach to constructing more effective computer tutors is implementing tutorial strategies that extend over multiple turns. This means that computer tutors must deal with (1) failure, (2) interruptions, (3) the need to revise their tactics, and (4) basic dialogue phenomena such as acknowledgment. To deal with these issues, we need to combine ITS technology with advances from robotics and computational linguistics. We can use reactive planning techniques from robotics to allow us to modify tutorial plans, adapting them to student input. Computational linguistics will give us guidance in handling communication management as well as building a reusable architecture for tutorial dialogue systems. A modular and reusable architecture is critical given the difficulty in constructing tutorial dialogue systems and the many domains to which we would like to apply them. In this paper, we propose such an architecture and discuss how a reactive planner in the context of this architecture can implement multi-turn tutorial strategies.

Motivation

Research on student learning has shown that students must construct knowledge themselves to learn most effectively (Chi *et al.* 1989; 1994). 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 (Cohen, Kulik, & Kulik 1982) and by 2.0 standard deviations with experienced tutors (Bloom 1984). 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 (Merrill, Reiser, & Landes 1992; Fox 1993; Graesser, Person, & Magliano 1995). Through collaborative dialogue, tutors can intervene to ensure that errors are detected and repaired and that students can work around impasses (Merrill *et al.* 1992). 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.
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 strategies, 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 BE&E developed with the VIVIDS authoring tool (Munro 1994). Students read textbook-style lessons written in HTML and then perform labs using the graphical user interface shown in Fig. 1. (Rosé *et al.* 2000) describes an experiment where students went through these lessons and labs with the guidance of a human tutor. The student and tutor communicated 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 strategies to be used by our tutor and plan to use them to train our system.

Our goal is to design a modular and reusable architecture that facilitates effective tutorial dialogue. The architecture must separate high-level tutorial planning (*e.g.*, teach procedure step-by-step) from low-level communication management (*e.g.*, all utterances must be acknowledged), allow the

*The research presented in this paper was supported by Grant #N00014-91-J-1694 from the Office of Naval Research, Cognitive and Neural Sciences Division.

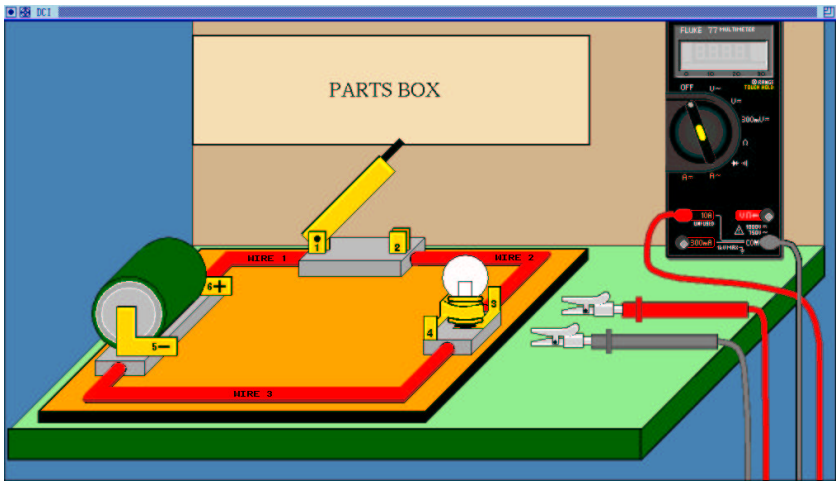

Figure 1: BE&E Graphical User Interface

design of a rich repertoire of domain-independent (portable) teaching strategies, define the knowledge sources involved in tutorial planning, and define a control strategy that separates the maintenance of knowledge sources from the planning process.

Previous Work

We evaluate the following four tutorial dialogue systems that support constructive learning: AutoTutor (Graesser *et al.* 1999) (domain: computer literacy), the CIRCSIM tutor with the APE dialogue planner (Freedman 2000) (domain: teaching doctors about the circulatory system), the Miss Lindquist algebra tutor (Heffernan & Koedinger 2000), and the EDGE system (Cawsey 1989) (domain: explaining various electric circuits). We evaluate these systems along three dimensions: (1) What type of tutorial strategies does the system contain? (2) What type of planner is used to combine these strategies into a plan? (3) Is low-level communication management separated from high-level tutorial dialogue planning?

Type of tutorial strategies used

We analyze the use of teaching strategies in the aforementioned tutorial dialogue systems along three dimensions: *extent*, *generality*, and *representation*. By extent, we mean: do the tutorial strategies extend solely over single turns, multiple turns, or both? A teaching strategy is specific to a domain if it does not separate tutorial knowledge from domain knowledge (*i.e.*, it contains hard-coded domain knowledge). A teaching strategy is general if its body contains abstract domain references that can be informed and instantiated by a domain reasoning mechanism. Unlike domain specific teaching strategies, general teaching strategies can be easily ported to another domain provided that the new domain provides similar domain reasoning mechanisms. A third dimension of teaching strategies is their representation, *e.g.*, as production rules, or action descriptions (operators).

AutoTutor contains only single-turn teaching strategies called *dialogue moves*. AutoTutor has ten dialogue moves: three short feedback moves for providing positive, neutral and negative feedback; and more substantive moves such as: "pumping", "prompting", and "hinting".

To discuss the generality and representation of the teaching strategies we need to introduce the notion of *curriculum script*. The curriculum script is a sequence of *topic formats*, each of which contains a *main focal question*, and an *ideal complete answer* hard coded in English. The ideal complete answer consists of several sub-answers, called *aspects*. Each aspect has the following slots: a list of anticipated bad answers corresponding to misconceptions and bugs that need correction (with splicing/correcting moves); lists of prompts and hints that can be used to get the learner to contribute information; and elaboration and summary moves that can be used to provide the learner with additional or summarizing information. Note, all of the moves are hard coded in English.

APE provides single-turn (*e.g.*, "primary-vbl-incorrect-3") and multi-turn (*e.g.*, "do-neural-DLR") teaching strategies. These example strategies are representative of APE's domain dependent library of strategies. It is therefore difficult (in contrast to AutoTutor) to cluster them into classes.

In APE, a teaching strategy is represented as a data structure called an operator which consists of several slots. The *goal* slot is achieved if the operator is successfully executed; the *precond* slot contains a number of constraints that must be true for an operator to be applicable; and the *recipe* slot contains a number of sub-goals that are generated by applying the operator.

EDGE is an explanation system in which students are periodically tested as well as allowed to ask questions at predefined points during explanations. EDGE has single turn teaching strategies as well as strategies that can unfold over multiple turns (*e.g.*, "explain how-it-works") and *remediation plans* that can deal with specific types of wrong answers.

EDGE provide two types of (STRIPS-like) operators: *content* and *discourse* operators. Content operators specify how to explain something. For example, "to describe a device: explain its function, its structure, and its behav-

ior". Discourse operators are used for communication management and will be explained below. EDGE's content operators are quite general; their bodies contain abstract domain references that interface with a knowledge representation module.

Miss Lindquist provides single-turn strategies (*e.g.*, hints), and multi-turn strategies (*e.g.*, "concrete articulation", "substitution with decomposition"). For defining complex strategies, Miss Lindquist introduces tutorial questions of different kinds, for example, "Q_compute" (find a numerical answer), "Q_explain" (write a symbolization for a given arithmetic quantity), and "Q_generalize" (use the result of Q_explain abstractly). The concrete articulation strategy consists of the sequence: Q_compute, Q_explain, and Q_generalize. Naturally, some question types are specific to the domain, e.g., "Q_order_of_ops" or "Q_substitute". Also, there is a strategy exclusively dealing with parentheses errors.

Hint chains are associated with questions and represented as lists of hard coded rephrasings of the question in English. A complex strategy is represented as a list of question types to be asked. At the time of writing, it is unclear how questions are represented.

Type of planner used

Planning tutorial dialogue is complex. When a tutor constructs a plan it does not have an accurate model of the student. The tutor may plan to teach a student some information but later find out that the student knows it. In examining the BE&E dialogues we have seen that the human tutor is careful not to perform steps of a teaching strategy that are unnecessary. For example, 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. The difficulty in implementing such behavior is making sure the resulting discourse is coherent when parts are left out.

Other occasions where a plan may need to be modified occur when a student asks a question, gives an incorrect answer, or gives a correct answer with minor errors unrelated to the main point of the question. If the student gives an incorrect answer, the tutor must immediately get the student "back on track". The two other cases introduce tangential goals (answer student question and correct minor errors). The planner must decide whether to deal with these tangential goals immediately, deal with them later, or dismiss them as irrelevant. Evidence from our corpus shows that tutors do not always deal with tangential goals immediately, and instead choose to focus on more serious errors first. In general, always dealing with tangential goals as they arise could lead to an unfocused dialogue with many digressions. Many digressions could also mean that the tutor does not meet its teaching goals (it runs out of time) and the student gets frustrated (if the tutor constantly corrects small errors).

Before discussing AutoTutor, APE, EDGE, and Miss Lindquist, it is worth considering the three-level planning architecture used in robotics (Bonasso *et al.* 1997). The first level consists of executing physical skills such as grasping a handle. The second level consists of executing sequences and monitoring their success. Despite their name, sequences can be more than lists of skills and can contain conditional statements, loops, and other sequences. The third level, deliberative planning, decides what sequences to pass to the second level.

From the perspective of tutorial systems, deliberative planning takes tutorial goals and develops plans (sequences) from its library of tutoring strategies. The second level executes these plans and monitors plan execution. Actions are executed by passing control to the skill level where, for example, a feedback generator decides how to realize a tutor move in natural language. If the second level detects that the plan has been interrupted or has failed then it calls the deliberative planner to modify the plan to deal with the disruption. We will now describe specifically how this three level architecture relates to AutoTutor, APE, EDGE and Miss Lindquist.

AutoTutor operates on the first and second planning levels. After having asked a question, AutoTutor evaluates the student's answer against all the aspects of the ideal complete answer, and the anticipated bad answers. AutoTutor gives immediate feedback based on the student's answer, and then executes dialogue moves that get the learner to contribute information until all answer aspects are sufficiently covered. Selecting these dialogue moves is the task of the dialogue move generator which takes into account: (1) the quality of the student's assertion in the preceding turn; (2) global parameters like student ability, verbosity and initiative; and (3) the extent to which the good answer aspects have already been covered. A set of 20 fuzzy production rules determines the category of the dialogue move to be selected. The content of the dialogue move is computed by an algorithm that selects the next good answer aspect to focus on.

In terms of the 3-level robotics architecture, AutoTutor's mechanism could be described as a complex sequence being executed in the second level. The content computation could also be viewed as an elementary tutorial skill, and be placed in the first level. Having no multi-turn strategies (and the complexities they add to tutorial dialogue planning), AutoTutor does not (need to) act on the third planning level.

Miss Lindquist also has a predefined second level of sequences. Sequences such as concrete articulation consist of skills (tutorial questions), in this case: Q_compute, Q_explain, and Q_generalize. The questions appear to be implemented in the skill level as simple templates; The sequence interpreter is more sophisticated, and handles cases where the student gives more information than requested, and answers a question that was due to be asked.

EDGE incrementally builds and executes new sequences. Before each tutor turn, the deliberative planner expands the current unfinished sequence by adding new sequences until it adds a skill. The first level then executes this skill using simple template driven generation. Thus, planning is delayed as much as possible so that the most current student model can be consulted. The sequences of EDGE can consist of subsequences as well as conditional statements: if a student does not know X, then teach X.

APE also incrementally constructs and executes se-

quences, and uses simple template driven generation in its skills level. However, APE conflates the second and third levels by embedding control in operators, unlike traditional planners, where control is separated from action descriptions (operators). This makes writing operators difficult and puts an additional burden on the planner. Thus, we advocate not allowing sequences to make control decisions.

Separation of communication management and tutorial planning

Communication management refers to actions that explicitly help maintain contact, perception, and understanding during the dialogue (e.g, acknowledgments, clarifications, confirmations). It includes basic conversational principles for cooperative dialogue partners: questions have to be answered; answers have to be acknowledged. Tutorial planning is concerned with constructing effective tutorial dialogue from a set of teaching strategies.

AutoTutor uses its three feedback moves to handle low-level communication management. After each student turn, the tutor gives immediate feedback (positive: "That's right,", "Yeah."; neutral: "Uh-huh"; negative: "Not quite,", "No.") preceding its more substantive dialogue moves.

APE conflates low-level communication management and high-level dialogue management into its operators. They can contain canned text schemas like

```
'No, actually' ?max 'controls' ?partial '.'
```

containing negative feedback, the discourse cue "actually", and a hint or answer move.

The authors of **Miss Lindquist** regard communication management operations (*e.g.,* generating English discourse cues) as tutorial operations.

EDGE makes a step towards separation by encoding tutorial knowledge in content operators and communication management knowledge in discourse operators. EDGE uses Sinclair and Coulthard's 4-level hierarchical model of classroom instruction (Sinclair & Coulthard 1975):

```
transactions - teaching exchanges - moves - acts.
```

Discourse operators are used for implementing both high-level structure and low-level communication management acts. They control interactions with the user and define how to enrich the discourse at the transaction level with meta-comments ("I will now explain to you how X works"), and on the move level with discourse markers ("Right", "OK").

Discussion

To summarize, some previous and ongoing work in tutorial dialogue systems has striven to support unconstrained natural language input and extended tutorial strategies but this work has had the following limitations: teaching strategies are domain-specific (APE, AutoTutor); the set of tutorial strategies is small (EDGE, Miss Lindquist); some systems embed control in plan operators (APE); all current tutorial dialogue systems except EDGE mix high-level tutorial planning with low-level communication management; and planning is conflated with student modeling and maintenance of the dialogue context (APE, EDGE). These limitations can make a system less maintainable, extensible, and portable.

It is also worth considering dialogue systems not designed for tutoring (Allen *et al.* 2000; Pieraccini, Levin, & Eckert 1997; Lewin 1998; Larsson *et al.* 2000; Rudnicky & Xu 1999; Chu-Carroll 1999). These systems do not allow for conversational moves extending over multiple turns and the resulting need to abandon, suspend, or modify these moves. However, these systems aim for dialogue strategies that are independent of dialogue context management and communication management concerns. These strategies contain no domain knowledge; they query domain reasoners to fill in necessary details. Furthermore, in systems explicitly performing dialogue planning, control is never embedded in plan operators.

Our goal is to combine these beneficial features (modularity and reusability) with the flexibility and educational value of tutorial systems with reactive planners. In the next section, we present a modular tutorial dialogue system architecture and show at a high level how this architecture and its reactive planner would handle a human-human tutorial dialogue.

Proposed Architecture

The appendix depicts the start of a BE&E dialogue. Utterances labeled GUI are text displayed as part of the graphical user interface while utterances labeled S (Student) and T (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 also shown in the appendix. Preconditions are depicted as lines connecting actions.

The BE&E 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 incorrect 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 our modular architecture with its reactive planner can be used to deal with these unexpected responses.

Our proposed tutoring architecture is shown in Fig. 2. Control flows through the architecture as follows:

1. When the student types input, the parser produces an underspecified logical form that the interpreter attempts to fully specify.
2. The interpreter uses the problem solving manager, dialogue context, current goals, 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 sends 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. The dialogue planner decides how to react to messages from the interpreter.

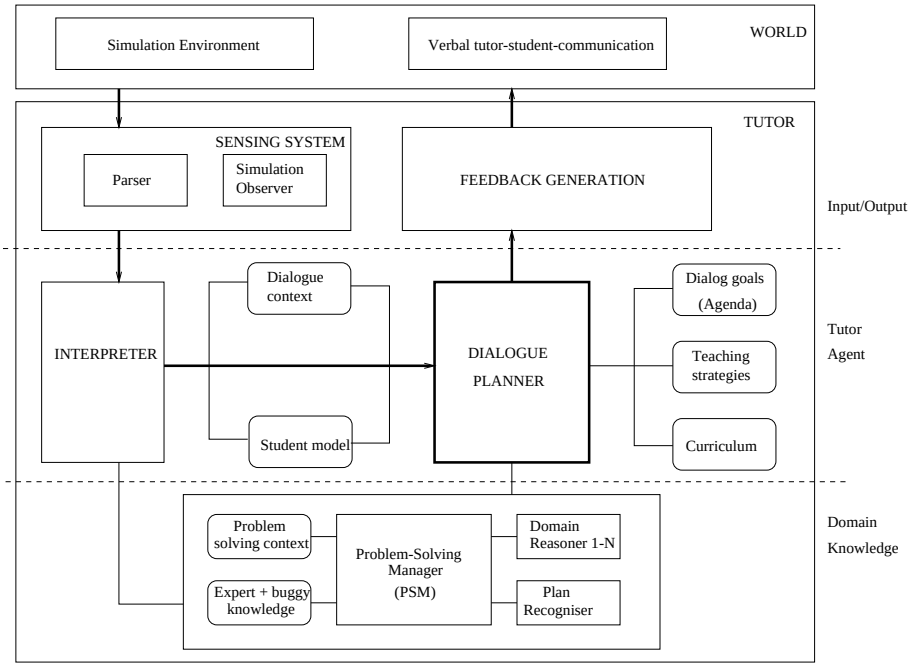

Figure 2: The BE&E Tutor Architecture

6. The feedback generator produces text from the dialogue move(s) output by the dialogue planner. Student model and dialogue context are updated.

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 to perform this action. We are assuming a probabilistic student model such as the one presented in (Conati *et al.* 1997) 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 this section, we describe how a deliberative planner could adapt plans to deal with student input. We are not claiming that this is the way to divide responsibilities between the planning levels, as this is an open research issue. Our goal is merely to provide an idea of the flexibility required to implement multi-turn tutorial strategies. Consequently, we also do not discuss communication management here. See the discussion section for our thoughts on communication management.

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 multi-turn teaching tactic? Here, we make the same decisions as the human tutor.

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 deliberative planner's agenda (a data structure for goal storage). In this case, the goal is `(student-performs (measure current))`.

The deliberative planner constructs a plan for achieving this goal using operators from its domain independent plan library. The library contains both single and multiple turn tutorial strategies. None of the operators contain domain knowledge; instead they contain abstract domain references. Consider the TEACH-STEP-BY-STEP operator (see Fig. 3) which states that to perform an action a student must perform all the substeps of that action. Substeps of particular actions are not encoded in the operator. Rather, the function PSM-ASK DECOMPOSITION retrieves the substeps from the domain reasoner.

The operators relevant to utterances 1-19 are shown in Fig. 3. Note the prefix *s-* in operators such as *s-knows* refers to student. The operators contain precondition and constraint slots that must be true before the operator can be executed. Constraints differ from preconditions in that constraints cannot be made true through the application of other operators. Unlike a STRIPS operator which is implicitly linked to one action, our operators are linked to zero or more actions as specified in their action slot.

To simulate the high-level structure of the dialogue in the appendix, we use the multi-turn TEACH-STEP-BY-STEP operator. When adding an operator to a plan, we first push any actions of the operator onto the agenda. In this case, there are none. Next the preconditions of the operator are added to the agenda. In this case, the preconditions are that the student performs the substeps of measuring current (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.

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


TEACH-STEP-BY-STEP ?a (multi-turn strategy)

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

TEACH-NEXT-STEP ?a (multi-turn strategy)

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

PRIME-NAME-NEXT ?a (multi-turn strategy)

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

ASK ?a (single-turn strategy)

effects:  (s-knows ?a) (s-states ?a) (salient ?a)
precondition: (primed ?a)
action: (ASK-MOVE ?a)

INSTRUCT ?a (single-turn strategy)

effects: (s-performs ?a)
action:  (INSTRUCT-MOVE ?a)

INSTRUCT2 ?a (single-turn strategy)

effects: (s-performs ?a) (s-performs ?b)
action:  (INSTRUCT-MOVE ?a ?b)
```

Figure 3: Dialogue Planning Operators

To address subgoal A, the planner chooses the INSTRUCT operator which has no preconditions and one action, uttering an INSTRUCT dialogue move conveying that the student should de-energize the circuit. An INSTRUCT move simply means to give an instruction. To implement skills involving communicative actions, the feedback generator produces natural language text given a move and the move's content. In this case, the generator is given the INSTRUCT move and the content `(de-energize circuit)` and produces "Set the switch (i.e., the circuit switch) to off".

In the example dialogue, the student flips the switch and goal A is popped from the agenda. Goals B and C on the agenda are addressed successfully in the same manner.

The planner addresses goals D and E using the INSTRUCT2 operator.[1] The INSTRUCT2 operator has the action of producing the INSTRUCT move seen in utterance 8. In response, the student removes the wire and the planner pops goal D from the agenda. The tutor waits for the student to connect the leads but eventually the time since the student last performed an action exceeds some tutor set threshold and the tutor decides to speak. The deliberative planner must now try an alternative strategy for getting the student to connect the leads.

We simulate the teaching strategies 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) and how to perform it (goal E2) in order to execute the step (see agenda below). A precondition of asking the student to identify the next step (goal E1.2) is priming the student (goal E1.1) which is done through PRIME-NAME-NEXT. 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. TEACH-NEXT-STEP is a fairly simple example of a directed line of reasoning (Hume *et al.* 1996).

```
AGENDA:

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

Some goals on the agenda are tied to the student model as suggested by (Cawsey 1989). 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. 3 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.

ASK-MOVEs simply ask questions, and the ASK-MOVE associated with E1.1.1.1 is realized by the question in utterances 10-11. After the question is asked E1.1.1.1 is popped off the agenda; the student makes the reply seen in utterance 12. Since the student answered correctly, the planner

[1] 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.

pops E1.1.1 off the agenda. The same scenario occurs for utterances 14-16 and 17-19 (ASK-MOVEs are answered correctly) 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": the interpreter with the help of the BE&E domain reasoner determines that (1) "one of the wires on the right of the picture" is vague and can refer to either lead; (2) the curriculum dictates that student should use the term "lead" instead of "wires"; (3) the answer is incomplete (it does not say where to attach the first lead or anything about the other lead). The interpreter encodes these problems as possible 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.

In utterance 27 ("do you understand why you are doing that?"), the tutor expects the student to say something like "to measure current" but instead the student says "because you need to connect the red wire to the beginning of the wire" which is basically a rephrasing of the question. To deal with this answer, the tutor repeats the question: "Why?". However, the student again simply rephrases the question. The tutor needs to realize that it has already tried repeating the question and must try something new. In fact, the tutor switches strategies and instead of asking for the goal of the action, asks for the goal of the lab. Notice the techniques used here apply to any complex action to be performed by a student not just connecting leads.

Discussion

In the section above, we have conflated the second and third levels in the robotics planning architecture. The deliberative planner should only operate on the third level, building sequences and adding sequences to the second level. Plan execution should only be performed in the second level. The first level of the planning architecture should only execute skills. The feedback generator gets a symbolic representation of a move (say hint, prompt, instruction) and performs the skill of constructing the corresponding utterance.

The second level may be an appropriate location to handle communication management. Communication management could be defined by rules such as: if the student has not responded after some threshold, then the tutor should speak; if the student responds, then the tutor must acknowledge the student; acceptance means acknowledgment.

To implement our system, we are using the TRINDI framework (Larsson & Traum 2000). Designing a TRINDI dialogue manager consists of four steps: defining (1) an information state (IS), (2) a set of dialogue moves (the communicative functions of utterances), (3) a set of update rules which update the IS taking the last dialogue move(s) and the IS into account, (4) and a control strategy which defines how to apply update rules. The IS can be thought of as the mental state of a dialogue participant, and contains conversational goals and information gained in the dialogue. External reasoning components (*e.g.*, a domain reasoner) communicate with the dialogue manager by reading and writing the IS.

To translate the architecture presented above into the TRINDI framework, we must first divide the IS into independent substructures (*e.g.*, the student model, problem solving context, and agenda). This should make the resulting system more portable and easier to maintain and extend.

Dialogue moves used in current TRINDI systems are relatively simple (ASK, ANSWER, ACCEPT, etc.). We are analyzing the BE&E dialogues, and investigating using dialogue moves such as WRONG-ANSWER and NEAR-MISS in our system.

We can build upon previous TRINDI systems that have implemented a large set of update rules for low-level communication management and simple dialogue planning. Different TRINDI systems such as EDIS (Matheson, Poesio, & Traum 2000) and Godis (Larsson *et al.* 2000) have chosen to focus on different concerns. EDIS has a comprehensive mechanism for handling low-level communication management using the notion of obligations. Godis focuses more on higher-level dialogue planning. However, EDIS and Godis work on very simple information seeking domains, and their update rules must be augmented with tutorial strategies. We are analyzing the BE&E dialogues to develop a set of domain independent tutoring strategies for our tutor.

EDIS and Godis have very simple control strategies. For example, EDIS clusters update rules into rule classes and given an IS and dialogue move collects all applicable rules of all rule classes and executes them one by one. Godis' control strategy features simple planning (*task accommodation* — the system will load and execute plans to achieve user goals) and replanning (*question accommodation* — when a user provides more information than required, the system acts as if it had asked for that information). Because of the complexity of tutorial planning, we are investigating extending Godis' simple accommodation strategies into a fully fledged three level planning architecture.

By building a dialogue system within the TRINDI framework we can not only profit from the framework's modularity but also from the work done on the currently implemented TRINDI systems. Thus, we are brining together technology from three areas: reactive planning from robotics, basic dialogue system technology from computational linguistics, and expertise in teaching from the ITS community. Using these knowledge sources, we believe that we can provide a modular architecture that maintains separa-

tion between domain knowledge, tutorial planning, communication management, student modeling, and maintenance of the dialogue context. This should increase the reusability, extensibility, and maintainability of the resulting tutorial dialogue system. Such a tutor with its reactive planner will also have flexibility: it will be able to abandon or modify failing teaching strategies, skip steps in a plan that have already been achieved, and deal with unexpected answers and interrupting questions. We believe this architecture enables some of the capabilities necessary for a tutor that supports constructive learning, in particular, supporting multi-turn teaching strategies.

References

Allen, J.; Byron, D.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2000. An architecture for a generic dialogue shell. *Natural Language Engineering*. Upcoming in the Special issue on Best Practices in Spoken Language Dialogue Systems Engineering.

Bloom, B. S. 1984. The 2 Sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. In *Educational Researcher*, volume 13, 4–16.

Bonasso, R. P.; Firby, R. J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental Theoretical Artificial Intelligence* 9:237–256.

Cawsey, A. 1989. Explanatory dialogues. *Interacting with Computers* 1(1):69–92.

Chi, M. T. H.; Bassok, M.; Lewis, M. W.; Reimann, P.; and Glaser, R. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science* 13(2):145–182.

Chi, M. T. H.; Leeuw, N. D.; Chiu, M. H.; and Lavancher, C. 1994. Eliciting self-explanations improves understanding. *Cognitive Science* 18(3):439–477.

Chu-Carroll, J. 1999. Form-based reasoning for mixed-initiative dialogue management in information-query systems. In *Proceedings of the 7^{th} European Conference on Speech Communication and Technology*, 1519–1522.

Cohen, P. A.; Kulik, J. A.; and Kulik, C. C. 1982. Educational outcomes of tutoring: A meta-analysis of findings. *American Educational Research Journal* 19:237–248.

Conati, C.; Gertner, A.; VanLehn, K.; and Druzdzel, M. 1997. On-line student modeling for coaching problem solving using bayesian networks. In Jameson, A.; Paris, C.; and Tasso, C., eds., *User Modeling: Proceedings of the Sixth International Conference, UM97*, 231–242. Springer.

Fox, B. A. 1993. *The Human Tutorial Dialogue Project: Issues in the design of instructional systems*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Freedman, R. 2000. Using a reactive planner as the basis for a dialogue agent. In *Proceedings of the Thirteenth Florida Artificial Intelligence Symposium (FLAIRS '00)*.

Graesser, A. C.; Wiemer-Hastings, K.; Wiemer-Hastings, P.; Kreuz, R.; and Tutoring Research Group, University of Memphis. 1999. AutoTutor: A simulation of a human tutor. *Cognitive Systems Research* 1:35–51.

Graesser, A. C.; Person, N. K.; and Magliano, J. P. 1995. Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology* 9:495–522.

Heffernan, N. T., and Koedinger, K. R. 2000. Building a 3^{rd} generation ITS for symbolization: Adding a tutorial model with multiple tutorial strategies. In *Proceedings of the ITS 2000 Workshop on Algebra Learning, Montreal, Canada*.

Hume, G.; Michael, J.; Rovick, A.; and Evens, M. 1996. Hinting as a tactic in one-on-one tutoring. *The Journal of the Learning Sciences* 5(1):23–47.

Larsson, S., and Traum, D. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. Submitted for publication. Available at http://www.ling.gu.se/ sl/papers.html.

Larsson, S.; Ljungloef, P.; Cooper, R.; Engdahl, E.; and Ericsson, S. 2000. GoDiS - an accommodating dialogue system. In *Proceedings of ANLP/NAACL-2000 Workshop on Conversational Systems*.

Lewin, I. 1998. Autoroute dialogue demonstrator. Technical Report CRC-073, SRI Cambridge.

Matheson, C.; Poesio, M.; and Traum, D. 2000. Modelling grounding and discourse obligations using update rules.

Merrill, D. C.; Reiser, B. J.; Ranney, M.; and Trafton, J. G. 1992. Effective tutoring techniques: Comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences* 2(3):277–305.

Merrill, D. C.; Reiser, B. J.; and Landes, S. 1992. Human tutoring: Pedagogical strategies and learning outcomes. Paper presented at the annual meeting of the American Educational Research Association.

Munro, A. 1994. Authoring interactive graphical models. In de Jong, T.; Towne, D. M.; and Spada, H., eds., *The Use of Computer Models for Explication, Analysis and Experimental Learning*. Springer Verlag.

Pieraccini, R.; Levin, E.; and Eckert, W. 1997. AMICA: The AT&T mixed initiative conversational architecture. In *Proceedings of the 5^{th} European Conference on Speech Communication and Technology*.

Rosé, C. P.; Moore, J. D.; VanLehn, K.; and Allbritton, D. 2000. A comparative evaluation of socratic versus didactic tutoring. Technical Report LRDC-BEE-1, LRDC, University of Pittsburgh.

Rudnicky, A., and Xu, W. 1999. An agenda-based dialog management architecture for spoken language systems. In *International Workshop on Automatic Speech Recognition and Understanding (ASRU), Keystone, Colorado*.

Sinclair, J. M., and Coulthard, R. M. 1975. *Towards an Analysis of Discourse: The English used by teachers and pupils*. Oxford University Press.

Appendix

```
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 cir-
cuit switch) to off.
S:   <turns off switch>
GUI: 3. Correct.
GUI: 4. Set the Multimeter to adc.
S:   <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.
S:   <selects wire 2>
GUI: 7. Correct.
GUI: 8. Remove the appropriate wire and connect
         the leads of the multimeter to the circuit.
S:   <only removes wire 2 from the circuit>
T:    9. OK,
     10. let's break this down into steps.
     11. What did the instructions ask you to do?
S:   12. remove the appropriate wire and connect
         the leads
T:   13. OK,
     14. and how much of this did you do?
S:   15. i only removed the wire
T:   16. Right.
T:   17. So what do you still need to do?
S:   18. connect the leads
T:   19. Right.
     20. And how are you going to do that?
S:   21. pick one of the wires on the right of the picture
T:   22. You mean the leads of the multimter?
S:   23. yes
T:   24. Which one will you pick and where will you
          attach it?
S:   25. i believe that I'll pick the red wire and
          attach to where I removed the beginning
          of the wire
T:   26. OK,
     27. do you understand why you are doing that?
S:   28. because you need to connect the red wire
          to the beginning of the wire
T:   29. Why?
S:   30. i'm not exactly sure.
     31. i just thought that you connect the red
          wire to the beginning
T:   32. Do you understand what the goal of this lab is?
S:   33. to be able to connect the multimeter to the
          light and read the number of amps are flowing
T:   34. OK,
     35. in order for any amps to be flowing, what has
          to be true about the circuit?
S:   36. it has to be  complete
T:   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)?
S:   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
T:   41. Good.  You are absolutely correct.
     42. You need to do both of those things before you
          press continue.

[...]
```

Figure 4: A BE&E Dialogue on Measuring Current

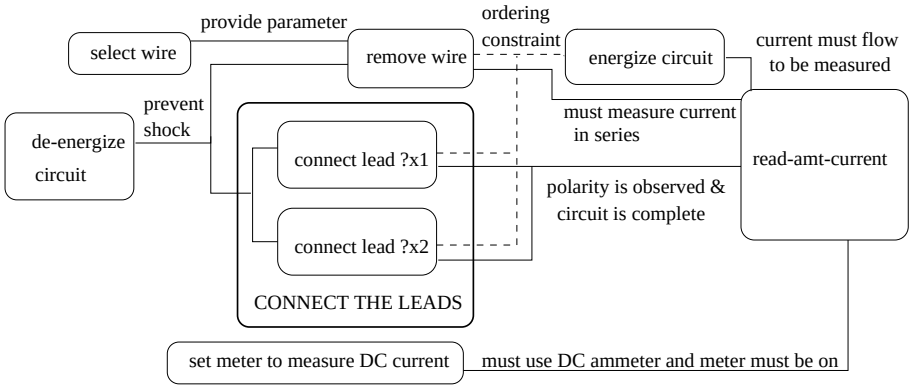

Figure 5: The BE&E Domain Plan for Measuring Current