# Using a Domain-Independent Reactive Planner to Implement a Medical Dialogue System


**Reva Freedman**
**Nicholas Haggin**
**Dessislava Nacheva**
**Tom Leahy**
**Richard Stilson**

Department of Computer Science
Northern Illinois University
DeKalb, IL 60115

freedman@cs.niu.edu
http://www.cs.niu.edu/~freedman



### Abstract

APE is a domain-independent reactive planning system used to build dialogue systems. Turns in the dialogue may include graphical actions as well as typed text. While APE runs fast and is easy to use, it is more powerful than many other dialogue managers because it can handle multi-turn plans, update plans during execution, and generate nested discourse constructs. Although APE is intended for creating dialogues containing hierarchical, multi-turn plans, it can be used to implement any dialogue system. In this paper we describe the motivation for the APE system, its internals and its uses. We illustrate the use of APE with CAPE, an APE-based system that tutors medical students on blood pressure regulation.


## Introduction

Dialogues play an important role in interactive applications involving communication between a person and a computer, such as intelligent tutoring systems (ITSs), advice-giving systems and interactive help systems. Although these types of systems differ in many features, such as where the locus of control resides, each of them can fundamentally be considered as a dialogue system.

APE is a domain-independent dialogue manager that conducts mixed-initiative, multimodal conversations with a human user. By multimodal we mean that both participants can use typed text and/or GUI actions. Conversations with APE-based systems can include nested multi-turn dialogue plans, such as a series of questions or questions and statements. APE can change plans during execution to better match the user's needs.

A global planning engine is needed in order to ensure coherent dialogues and to ensure that the system can satisfy its goals during a conversation, but classical planning algorithms, which build a complete plan before execution, are inappropriate because the system must be able to change plans based on the user's responses. For this reason APE is based on ideas proposed by the philosopher Michael Bratman (1987, 1990) and elaborated in a computer science context by subsequent researchers (Bratman, Israel and Pollack 1988; Pollack 1992; Georgeff et al. 1998).

In this paper we describe some insights into human planning suggested by Bratman and others. Then we show how these insights have been incorporated into the reactive planning model. We describe how APE uses reactive planning to conduct a dialogue with a person. Finally, we give an example of an APE-based system, CAPE, that tutors medical students on blood pressure regulation.

## Comparison to other approaches

Dialogue systems are often implemented with finite-state machines, either simple or augmented. In the most common finite state model, each time the human user issues an utterance, the system reduces it to one of a small number of categories. For example, utterances may be classified as correct, incorrect, incomplete and so on. These categories represent the possible transitions between states. In a finite state machine, history and context can only be stored by expanding the number of states. There are several problems with this approach. First, it puts an arbitrary restriction on the amount of context or depth of conversational nesting that can be considered. More importantly, it misses the significant generalization that goal-oriented dialogues are hierarchical: larger units contain repeated instances of the same smaller units in different sequences and instantiated with different values.

Replacing the finite-state machine with a pushdown automaton might appear to produce a more flexible model. But pushdown automata generate context-free grammars, i.e. once a subdialogue with a particular decomposition has been started, there is no way to terminate it before all of its steps have been realized in text. This approach is not flexible enough for realistic dialogues since it does not allow the computer agent to change the course of the dialogue based on the human agent's responses. If the computer agent is working its way through a series of questions, either to obtain information or to produce a Socratic dialogue, it is preferable to terminate the series as soon as it becomes clear that it the remaining questions are no longer required.

In addition, both the finite-state and pushdown automaton models assume that user input can be mapped onto a small number of categories. Although APE can handle sets of categories, it can also handle input expressed directly as propositions with arguments, thus allowing a much finer-grained input representation.

## Learning from human planning

### Bratman's "Practical Reason"

Bratman uses the term "practical reason" to describe his analysis since he is concerned with how to reason about practical matters. For human beings, planning is required in order to accomplish one's goals. For a dialogue system, planning is required both to ensure a coherent conversation as well as to ensure that the system accomplishes its goals. But it is impossible to plan a whole conversation in advance when the user can respond freely at every turn, just as human beings cannot completely plan their daily lives in advance because of possible changes in conditions. In short, one cannot fully plan a conversation in advance because it is impossible to predict what the other agent is going to say.

On the other hand, human beings do not decide what to do from scratch at each turn. People make plans and tend to follow a plan once they have one. In other words, it is more efficient to have a plan and only change it when necessary. But people are capable of dropping a goal or changing a plan when necessary. To make this process more efficient, Bratman and others who have adopted his work leave plans at as high a level as possible until it is necessary to refine them.

### Implementation via reactive planning

Looked at from the system's point of view, a dialogue is partially planned and partially a reaction to the user's utterances. For example, in a tutoring system, the student starts a tutoring session by choosing a problem and starting to solve it. In general, when the student's response is correct, the tutor allows the student to keep working on the problem. When the student gets stuck or makes a serious error, however, the tutor may take the initiative. For example, the tutor might choose an easier problem for the student, give the student the answer, or add a remedial tutoring plan containing one or more steps. After executing the remedial subplan, the tutor may give control back to the student to continue solving the problem.

Reactive planning (Georgeff and Ingrand, 1989) is a technology that was developed for domains that need such integrated planning and execution. Originally developed for real-time control of the space shuttle, reactive planning has since been used in a variety of other domains including military transportation problems and industrial process control. We have developed a reactive planner called APE that uses these ideas to conduct a conversation. After each user response, the planner can choose to continue with its previous plan or change something in the plan to respond better to the user's utterance. Changes can include adding or deleting plan steps, or even changing from one subplan to another. In summary, we model conversation like a chess game: expert players have a plan for upcoming moves, but they are also capable of changing the plan when circumstances change.

### Hierarchical decomposition

Like most reactive planners, APE is a hierarchical task network (HTN) style planner (Yang 1990; Erol, Hendler and Nau 1994). Hierarchical decomposition asserts that each goal can be achieved via a series of subgoals. Decomposition is well-suited to large-scale dialogue planning because it is easier to establish the sequence of things a human speaker will say in a given situation than to be able to understand why in sufficient detail and generality to do means-end planning. Hierarchical decomposition also minimizes the search time to find matching plans in the plan library.

Additionally, task-oriented dialogues tend to have a hierarchical structure (Grosz 1977). Matching the structure of the domain simplifies operator development because operators can often be derived from transcripts of human dialogues. The hierarchy information can be used in operator preconditions as a way of maintaining coherence.

Finally, in a hierarchical decomposition system, interleaving planning and execution is straightforward and efficient because one can expand precisely the portion of the plan that is about to be executed. The conversation becomes in effect a trace of the planner.

## Structure of APE

### The APE planning algorithm

APE builds partial plans as trees. When an operator is chosen to decompose a node, the steps of the operator become subplans of the node and are represented as its children.

APE's high-level planning loop consists of the following steps:

1. Wait for input. APE runs in parallel with a user

interface, generally (but not necessarily) a graphical one (GUI). APE waits for user input from the GUI. Input can be in any form appropriate to the application, including menus, spreadsheet entries, text, or graphical objects. APE can also respond to computer-generated signals such as timeouts, which can be generated either by the GUI or internally. The author of the system is responsible for choosing a representation language to communicate between APE and the GUI. The GUI code interprets the user's input, whether text or graphical, and converts it into the representation language. When APE perceives a user input, it starts up at the point where it left off.

2. Decide how to respond. APE executes plan operators until a primitive, non-decomposable one is obtained. Each time a primitive operator is obtained, it is added to a buffer that will be used to create the system's response. This process ends when a primitive operator requiring a response from the user is added to the buffer. All decisions about how to respond to the user, including the content, the medium (text or graphics), and whether to retain control or cede it to the user, will be made through the use of plan operators.

3. Execute the turn. Depending on the application, the primitive operators may be sent directly to the user or may require further processing to build a coherent turn.

4. Return to step 1. The conversation is complete when both parties have completed their original goals or decided to drop any remaining subgoals.

## Structure of APE operators

APE stores its current partial plan in an agenda, implemented as a stack with additional operations available besides *push* and *pop*. The stack is threaded in order to represent the partial plan, which is a tree. The top of the stack represents the next goal to decompose. To initiate a planning session, the user invokes the planner with a goal, e.g. to conduct a conversation covering a specified topic, or to help a student solve a problem. The system stores the initial goal on the agenda, then searches the operator library to find all operators whose goal matches the top goal on the agenda and whose preconditions are satisfied. Goals are represented using first-order logic without quantifiers, with full unification used for matching.

Since APE is intended especially for generation of hierarchically organized task-oriented discourse, each operator has a multi-step *recipe* in the style of Wilkins (1988). When a match is found, the matching goal is removed from the agenda and is replaced by a series of subgoals representing the steps in the recipe.

Recipe items can take several forms:

- *Goals*. A recipe can be composed of a series of one or more subgoals. In later steps, additional recipes will decompose these subgoals.
- *Primitives*. The minimal set of primitives encompasses *inform*, used to send some information to the user, and *elicit*, used to request information from the user. The arguments to these primitives are defined by the system author. A built-in mechanism allows the system author to build a turn from concatenated strings with variable replacement without additional programming.

  To generate text, the system author should plan for a string of zero or more *inform*s followed by an *elicit*. This will generate a turn consisting of some (optional) information followed by a question or a command. In human-to-human dialogue, a turn can be purely informational, leaving it to the other person's implicit knowledge to know how to respond. However, in human-machine dialogue, if the system does not explicitly tell the user what is expected, the user is likely to assume a hardware or software failure.

  If desired, the GUI can process the response further before displaying it for the user. For example, if the primitives are sentences, the GUI might want to reformat them into a paragraph. Primitives can also be input to an external package. For example, to generate more elaborate textual answers, primitives could be input to a natural language generation package; in that case, the GUI would call the NLG package to build a textual response before sending it to the user. Alternatively, primitives could be calls to a database package, or statements in a user-defined language that the GUI would then use to generate graphics on the screen.
- *Knowledge base updates*. APE maintains an internal knowledge base that represents its "mind." Recipes can update the user's knowledge by adding or deleting facts from the knowledge base.
- *Run-time decisions*. APE can evaluate a condition and use the result to decide whether or not to skip the rest of the recipe. The type of run-time decision making is used to bypass the rest of an operator when circumstances change during its execution. It thus differs from preconditions, which are only checked when an operator is being chosen. Run-time decision making can be used to implement the control condition in a loop.
- *Updating the plan*. APE provides two ways to change a plan in progress. *Retry-at* implements a Prolog-like choice of alternatives. If there are multiple ways to satisfy a goal, *retry-at* allows one to choose among them a second time if one's first choice becomes undesirable. For *retry-at* to be useful, the author must provide multiple operators for the same goal, and each operator must have a set of preconditions enabling it to be chosen at the appropriate time. As in Prolog, *retry-at* can be used to implement a loop; one simply chooses the same operator every time until the exit condition is reached.

  *Prune-replace* allows a type of decision-making frequently seen in dialogue generation. When a conversation partner does not give the expected response, one would often like to pop the top goal from the agenda and replace it with one or more new goals. *Prune-replace* implements a generalized version of this concept, allowing one to pop the agenda until a desired configuration is reached.

## Knowledge used by APE

APE uses preconditions on plan operators to obtain information from the world as well as from its own data structures. The preconditions can access information from any available source. In a general way, this information can be divided into three categories:

- Information maintained by APE
- Information about interaction with the user
- Expert knowledge about the domain and the task

The first category includes information in the planning agent's "mind," i.e. information that would be in the mind of a human speaker. It includes the following items:

- Partial plan in progress
- Discourse history/interaction history

The partial plan expresses the agent's current goals. The discourse history is a record of the interaction with the user, including both text and GUI actions. Our current implementation includes information in the discourse history only when it is needed for further planning decisions.

The second category includes the following items:

- Interpretation of the student's latest utterance (GUI or natural language)
- Information about items visible on the screen

Although the model of what's on the screen, which is part of the current shared context, may be maintained by the host system, APE needs to access it in order to generate utterances that accurately match that shared context.

The third category is optional. The breadth and depth of domain information necessary depends on the goal of an individual system built with APE. Domain information can be used for making decisions, and it can also be communicated directly to the user.

## Data storage

The APE environment contains two knowledge bases. One contains permanent information such as domain facts, while the other contains assertions that are only true for some period of time, such as during the solution of one problem. Facts can be added and deleted from the transient knowledge base whenever desired.

APE also permits the user to declare *external relations*. These relations are used in the same way as other knowledge base relations, but they do not actually appear in the knowledge base. Instead, when an external relation is called, a user-written function is called to return the desired information in the same format as used by the knowledge base interface. The use of external relations allows the user to express preconditions which may be inconvenient to express in first-order logic (such as arithmetic), to connect

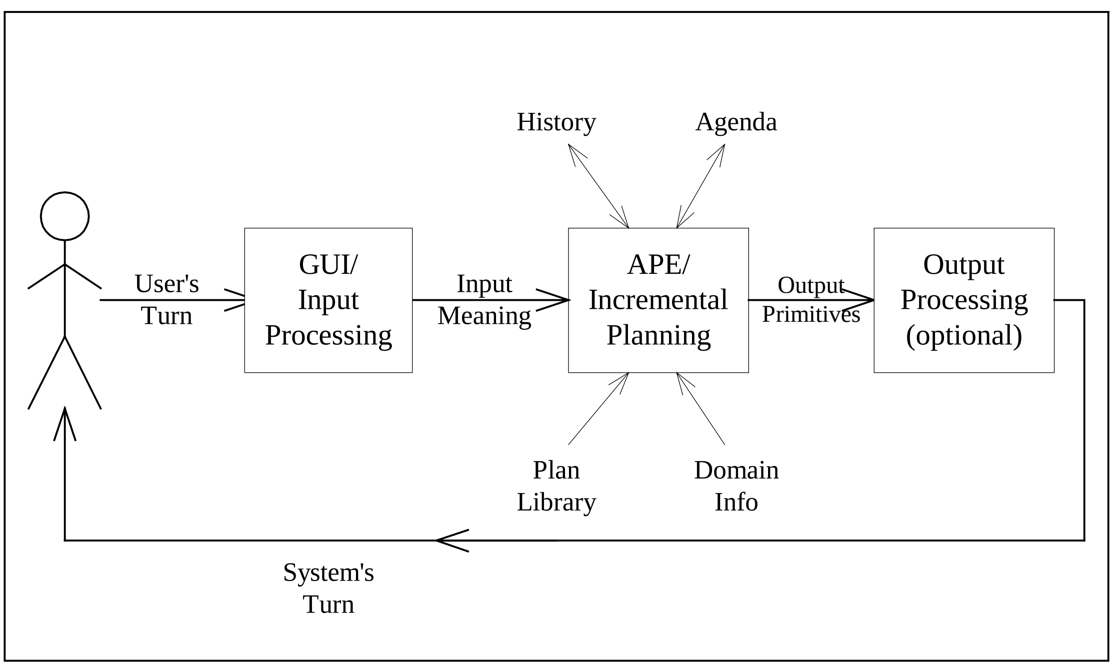


**Figure 1: Architecture of CAPE**

to an external database, or to communicate with another program, such as a domain expert.

The transient knowledge base is also used to help the planner communicates with other processes. Functions are provided for other processes to read and write facts in the knowledge base. In general, GUI information is transmitted to APE by storing it in the transient knowledge base.

Further details about the APE system can be found in Freedman (2000).

## CAPE: An APE-based System

We will use the CAPE system to illustrate the use of APE. The architecture of CAPE is shown in Figure 1. CAPE solves the same problem as the existing medical dialogue system CIRCSIM-Tutor v. 2 (Michael et al. 2003), but contains no code from that system.

### The CIRCSIM-Tutor Application

CIRCSIM-Tutor helps students master the negative feedback loop that maintains a steady blood pressure in the human body, one of the more difficult topics studied by first-year medical students in the Introduction to Physiology course.

Students are given a simplified qualitative model of the heart, followed by a series of problems which utilize the model. In each problem, an incident such as the administration of a drug affects the processing of the heart. Other incidents include broken pacemakers, transfusions and hemorrhages. The student is asked to predict the direction of change of seven core variables:

HR: Heart rate (beats/min)
TPR: Total peripheral resistance
IS: Inotropic state, a measure of the heart's contractile ability
SV: Stroke volume (vol/beat)

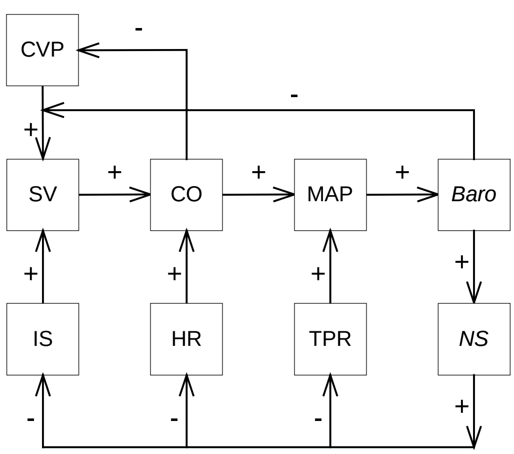


**Figure 2: Top-level concept map**

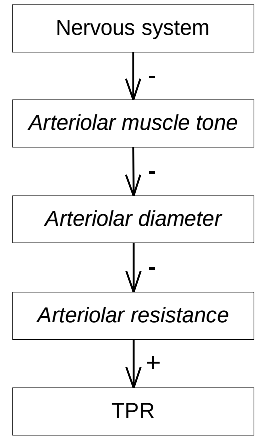


**Figure 3: Section of lower-level concept map**

CO: Cardiac output (vol/min)
MAP: Mean arterial pressure
CVP: Central venous pressure

The qualitative causal relationships between the core variables (i.e. an increase in X causes an increase/decrease in Y) are shown in Figure 2. In this diagram, NS = nervous system and Baro = the baroreceptors in the neck which recognize a change in blood pressure. A finer-grained knowledge representation is also available for the tutor to use when needed. A section of this knowledge base, the leg from the nervous system to TPR, is shown in Figure 3.

Each variable must be predicted at three points: the DR or *direct response* phase immediately after the incident, the RR or *reflex response* phase, which shows the effect of the nervous system, and the SS or *steady state* phase after a new steady state has emerged. After the predictions are made, the tutor engages the student in a dialogue to help the student learn the correct answers and the concepts underlying them.

The basic user interface is the screen shown in Figure 4. A description of the current problem is shown in the upper right. Below that is a table where the student can fill in predictions for the three stages. The left-hand side contains a window where the dialogue evolves.

The student part of the dialogue is free text. Instead of restricting what students can say, CAPE attempts to guide them toward understandable responses through the way the questions are framed, e.g. by asking short-answer questions instead of open-ended ones. CAPE uses rudimentary input techniques to see if an important keyword is used in the student's answer. Short-answer questions are preferred to multiple-choice questions because they trigger students' recall memory rather than just their recognition memory.

When CAPE does not understand a response, it uses the planning mechanism to generate the same type of response that a human being might use in similar circumstances. For example, if CAPE does not understand the student's

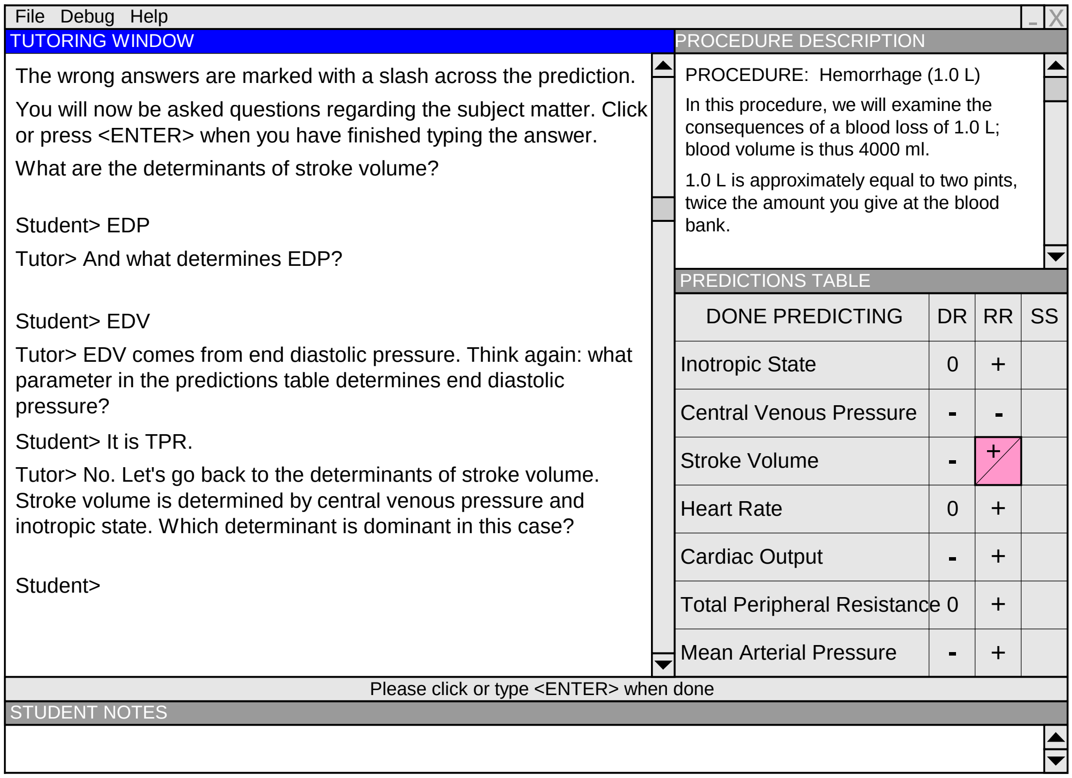


**Figure 4: CAPE user interface**

response to a question like "Then what happens to the heart rate?", it might respond by giving a list of possible alternatives: "Does it increase, decrease or stay the same?". The availability of the planning mechanism makes it possible to generate appropriate replies in context, rather than just a generic "I didn't understand you."

## Dialogues Generated by CAPE

At the top levels, the conversation generated by CAPE is fully hierarchical. Within each stage, as predicted by Grosz (1977), the text is divided into segments, one for each incorrect core variable. The variables are discussed in the sequence they are encountered in the solution of the problem.

Each variable is discussed using one of a number of multi-turn plans called *tutoring methods* that we have isolated from studies of human tutoring transcripts. The following example is typical:

To correct student's ideas about a variable
(Precondition: variable controlled by nervous system)
  Teach about variable's mechanism of control
  Teach about when this mechanism is activated
  Check whether student now knows the correct answer

Each tutoring method comprises one or more topics such as the three above. Unless it includes a recursive call to another method, each topic is decomposed into the standard text generation primitives *elicit* and *inform* with appropriate arguments. When the tutoring method is coded in APE, the list of topics will become the recipe for the plan operator. Preconditions are used so that different recipes are available depending on context. For example, the tutoring method above is only appropriate for variables controlled by the nervous system. Also, as a general policy, when the student does not understand something, CAPE tries a different tutoring method on its second try, just as a human tutor might. When a line of questioning has become unfruitful, CAPE always has access to the last-ditch tutoring method, namely giving the student the answer.

Instead of planning the complete text as in a monologue, APE interleaves planning and execution, planning only as much as necessary to generate the next turn. When the student gives an unexpected response, which includes various kinds of "near misses" as well as wrong answers, CAPE can choose between using a different method to decompose the current goal, adding a new goal at the top of the agenda, or dropping the current tutoring method and replacing it by another one. In this way it can respond flexibly to the student while still maintaining a long-range plan. As a result, at the lower levels of the plan, the conversation is no longer purely hierarchical.

For example, if the student gives the expected answer at every turn, the tutoring method above might generate the following conversation when the variable being discussed is total peripheral response (TPR):

T: What controls TPR?
S: The nervous system.
T: But the nervous system is not active in DR. That's the definition of the direct response stage. So now what do you think about the value of TPR?
S: It doesn't change.
T: Correct.

Many variants of this conversation can occur, depending on whether the tutor decides to teach a particular topic with a question, as for the first topic (mechanism of control of TPR), or with some information, as for the second topic (when this mechanism is activated). As another example of the type of flexibility provided by a plan-based system, note that CAPE does not mechanically say "right" or "wrong" after every student answer, but uses a policy closer to that of human tutors (Fox 1993). Also note that the second and third topics have been combined into one turn to make a smoother conversation. This is possible because in this example, the second topic just provides information and does not require any input from the student. Further details about the types of choices that CAPE can make can be found in Freedman et al. (2001).

Figure 5 shows the type of dialogue that CAPE would generate using the same tutoring method when the student gives comprehensible but unexpected answers. (Although the generated text is real, it should be noted that the dialogue is not realistic in the sense that genuine students do not usually make so many consecutive errors.) Near misses are answers that the tutor can use to build upon. It is incorrect, as well as unhelpful to the student, to label them as wrong, but the tutor does not want to let them pass without comment either. In the terms of Woolf (1984), they contain a "grain of truth."

The reader can see that the dialogue shown covers only the first topic of the tutoring method. Each time the student gives a near miss answer, the tutor adds one or more topics to the agenda. For example, in turn 3 the tutor asks an additional question to help the student relate the deeper knowledge in Figure 3 to the desired high-level answer, namely the nervous system. In turn 5, the tutor adds some additional information to the agenda before re-asking the previous question, which is still on the agenda. Eventually, in turn 10, the student gives a synonym of the correct answer. Since part of learning science is learning the appropriate vocabulary in context, the tutor accepts this response but sends out one final bit of information giving the exact answer desired before continuing with the next topic in the tutoring method.

Note that the tutor accepted the answer to the added question "What controls arteriolar radius?" in lieu of an answer to the original question "What controls TPR?". This implicit acknowledgment that the answers to the two questions are the same is a feature of the CAPE domain. To implement it, APE's plan-updating capabilities are used to drop the original question once a correct answer to the new question has been received. In another domain, this behavior might not be desirable. For example, in tutoring algebra, it might be useful to decompose the problem "How much money did Kim spend?" into two subtopics, "How many candy bars did Kim buy?" and "How much did each candy bar cost?". But the original question is still relevant and should be left on the agenda.

The intermediate parameters in Figure 3 represent deeper knowledge that the student does not need to invoke in order to solve the problem. But if the student does invoke that knowledge, the tutor wants to build on that information to obtain an answer to its original question. Notice the use of discourse markers such as "and" and "so" and narrative elements such as "try again" to keep the conversation on track. Providing this level of detailed feedback in a coherent conversation is not practical on a large scale without underlying software such as APE.

## Conclusion

APE is a reactive planning system that can be used to implement any conversation-based system, where turns in the "conversation" may include graphical actions and/or text. It can be used in any type of conversational domain, including tutoring, help systems and advice-giving systems. Different from finite-state systems, APE can easily model dialogues with hierarchical multi-turn plans. Unlike

1 T: What controls TPR?
2 S: Arteriolar radius. *(near miss)*
3 T: And what controls arteriolar radius?
4 S: Radius of arterioles. *(near miss)*
5 T: No, radius of arterioles and arteriolar radius are the same thing.
So what controls arteriolar radius?
6 S: Arteriolar diameter. *(near miss)*
7 T: Arteriolar diameter and arteriolar radius are the same thing in different units.
Try again. What controls arteriolar radius?
8 S: Arteriolar resistance. *(near miss)*
9 T: No, actually arteriolar radius controls arteriolar resistance.
So what controls arteriolar radius?
10 S: Sympathetic stimulation.
11 T: In other words, the nervous system.

**Figure 5: A complex CAPE dialogue**

pushdown automata, APE can change or abandon plan elements that are no longer relevant. APE makes it more practical to provide specific feedback for large numbers of user errors and unexpected responses. In this paper we illustrated these ideas with CAPE, a physiology tutor built using APE.

## Acknowledgments

We thank Martha Evens of the Department of Computer Science of the Illinois Institute of Technology and Joel Michael and Allen Rovick of the Department of Physiology of Rush Medical College, who were the principal investigators on the CIRCSIM-Tutor project.

## References

Bratman, M. 1987. *Intentions, Plans, and Practical Reason*. Cambridge, MA: Harvard.

Bratman, M. 1990. What is Intention? In P. Cohen, J. Morgan and M. Pollack, *Intentions in Communication*. Cambridge, MA: MIT Press.

Bratman, M., Israel, D. and Pollack, M. 1988. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence* 4(4): 349–355.

Erol, K., Hendler, J. and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI '94), Seattle, WA.

Fox, Barbara. 1993. *The Human Tutorial Dialogue Project*. Hillsdale, NJ: Lawrence Erlbaum.

Freedman, R. 2000. Using a Reactive Planner as the Basis for a Dialogue Agent. In Proceedings of the Thirteenth Florida Artificial Intelligence Research Symposium (FLAIRS 2000), Orlando.

Freedman, R. et al. 2001. Adaptive Processing in a Medical Intelligent Tutoring System. In Papers from the Workshop on Adaptation in Dialogue Systems, First Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2001), Pittsburgh.

Georgeff, M. and Ingrand, F. 1989. Decision-Making in an Embedded Reasoning System. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI '89), Detroit, MI, 972–978.

Georgeff, M., Pell, B., Pollack, M., Tambe, M. and Wooldridge, M. 1998. The Belief-Desire-Intention Model of Agency. In N. Jenning, J. Muller, and M. Wooldridge, *Intelligent Agents V*. Springer.

Grosz, B. 1977. The Representation and Use of Focus in a System for Understanding Dialogs. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI '77), Cambridge, MA, 67–76.

Michael, J., Rovick, A., Glass, M., Zhou, Y. and Evens, M. 2003. Learning from a Computer Tutor with Natural Language Capabilities. *Interactive Learning Environments.* 11(3): 233–262.

Pollack, M. 1992. The Uses of Plans. *Artificial Intelligence* 57(1):43-69.

Wilkins, D. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* San Mateo, CA: Morgan Kaufmann.

Woolf, B. 1984. Context-Dependent Planning in a Machine Tutor. Ph.D. diss., Dept. of Computer and Information Science, University of Massachusetts at Amherst. COINS Technical Report 84–21.

Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence* 6(1): 12–24.