Integrating Tools for the Creation of Speech-Enabled Tutors

Jonathan C. Brown
jonbrown@cs.cmu.edu

CMU-LTI-04-186

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

This report describes work done in enabling language-learning exercise creators to create speech-enabled intelligent tutor exercises. The utilities created make use of two existing sets of tools, the Cognitive Tutor Authoring Tools and the Sphinx-2 speech recognition system. The report describes how these tools are integrated, and how the resulting utilities can be used to create exercises that incorporate student speech. The reader is walked-though the creation of a simple speech-enabled exercise and introduced to more complicated exercises.

1 Introduction

Creating intelligent tutors, or even just tutor exercises, can be a time consuming and difficult process. However, much work is being done in making this quicker and easier. The goal of this project was to take advantage of this work and extend it to the field of language-learning. Specifically, this project was undertaken to extend the Cognitive Tutor Authoring Tools to be more useful for creating language-learning tutors in an LTI graduate course, "Language Technologies in Computer Assisted Language Learning" (11-717). This course was already using these authoring tools, but because speaking is a very important part of learning a new language, it was necessary to extend the tools to make it simple to use speech recognition functionality to build speech-enabled tutor exercises. In the next section, the existing tools used in this process are described. In the following sections, the desired abilities of a speech-enabled tutor, the utilities created to satisfy these desires, and the process of building a simple exercise are detailed. Finally, this report introduces some of the more complicated exercises that have been built with these utilities.

2 Existing Tools

There are two main toolsets that are used to aid in the development of these language-learning exercises. The first is the Cognitive Tutor Authoring Tools. The second is the open-source Sphinx-2 speech recognition system.

2.1 Cognitive Tutor Authoring Tools

The Cognitive Tutor Authoring Tools (CTAT) are a set of tools built by researchers at Carnegie Mellon University and Worcester Polytechnic Institute. They are designed to ease the development of intelligent tutors[1]. A subset of these tools allow for the creation of "Pseudo Tutors", which exhibit the normal behavior of intelligent tutors but do not require the creator to perform any AI programming[2]. In fact, authors build these tutors by simply creating the exercise interface and then demonstrating student behavior within that interface. The resulting tutor exercises simulate intelligent tutor behavior, such as the behaviors resulting from model tracing and knowledge tracing. Model tracing provides individualized assistance during problem solving through feedback and help messages, whereas knowledge tracing is used to estimate knowledge growth across problems[3, 4]. The pseudo tutor authoring process is designed to capture the visible effects of these processes as they occur in normal intelligent tutors[2].

The exact process of creating a pseudo tutor will be described later, when building the simple speech-enabled exercise. However, it is useful to take a look at the steps involved without the additional requirements for speech. The first step is to design the interface, using the tutor GUI builder provided in the CTAT[2]. This interface is made by dragging and dropping provided interface widgets onto the exercise window to create a Java interface. These widgets allow for a variety of different activity types. If desired, the user can also program this interface in code, because the user is really just creating a Java interface using special widgets. The second step is to demonstrate the correct and incorrect steps students may take when completing the exercise. This is done using the Behavior Recorder, also provided in the authoring tools[2].

The Behavior Recorder tool automatically creates behavior graphs, while the author demonstrates students' actions in the interface itself. These actions include both correct and incorrect actions. The author can also directly modify the behavior graph to mark certain paths as incorrect. In the third step, the author annotates the behavior graph further, adding help messages, error messages, and other feedback messages on different paths of the graph. This allows the author to specify the feedback the student receives when working on the problem. Finally, the author can annotate paths of the behavior graph with knowledge labels, which indicate specific skills used within the exercise.

Once these pseudo tutor exercises have been built, students can use them, and will receive intelligent feedback, based on their actions, as defined by the behavior graph. When students complete multiple exercises, the tools allow the author to view the performance of each student on each of the skills (denoted by knowledge labels), both within a single problem and across multiple problems that require some or all of the same skills.

The Cognitive Tutor Authoring Tools provide many resources for building full and pseudo intelligent tutors[1, 2]. Although the focus is on the tools used for pseudo tutors in this report, it should be mentioned that the utilities described later will also work with full tutors designed using the CTAT.

2.2 Sphinx-2 Speech Recognition System

Sphinx-2 is a real-time, large vocabulary, speaker-independent speech recognition system[5]. It is written in C, runs on UNIX and Windows, and is available under a free and open source license[6]. To recognize speech, Sphinx-2 must be provided with the following: an acoustic model, a language model, and a pronunciation lexicon.

Sphinx-2 comes with a fully trained acoustic model that can be used without modification if desired. The language model and pronunciation lexicon must be built and supplied to the system. One can easily create these using the online Sphinx Knowledge Base Tools[7]. This tool must be provided with a list of possible student utterances for this exercise. An utterance is a phrase or sentence that the student may be expected to speak. The online tool will use this sentence corpus file to build the necessary language model files and pronunciation lexicon. One can also add special words to the models if necessary; the CMU Pronouncing Dictionary may be useful for this[8]. See the walk-though example for more details.

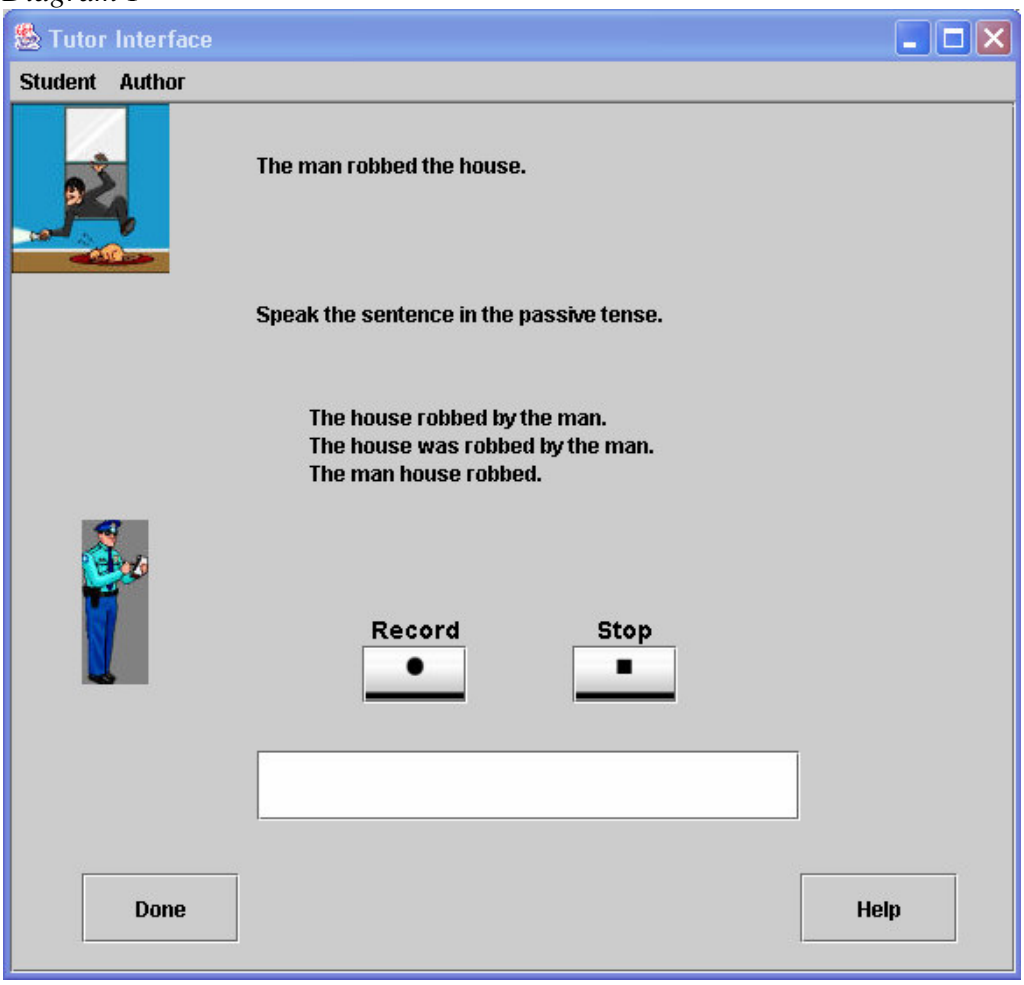
3 How a Speech-Enabled Tutor Works

This section describes exactly how one would like a speech-enabled tutor to work. That is, this section will describe the desired functionality of a completed tutor from the designer's perspective.

First, the student will be presented with a problem. This problem will require the user to speak, in order to give either an answer to a question, the next line of a dialog, or whatever else he or she is prompted for. To do this, the user will press a record button, speak their response,

and click the stop button. A recording of the user's utterance will then be passed off to the recognizer, which will already be set up to deal with this problem. The recognizer will then return one or more candidate utterances, which will be compared to the expected responses to determine the response the system believes the student gave. The learner will then receive feedback based on this determined response. Diagram 1 shows a completed tutor exercise with an interface designed to allow this functionality.

Diagram 1



4 Description of Tools Created

Given the existing tools, it was found that there were 2 main components necessary to provide the desired functionality of a speech-enabled tutor: an audio recording and recognition component and a tutor integration component. In this section, both of these components will be described. This set of tools, the main product of this project, is freely available online at [9].

4.1 Audio Recording and Recognition Component

The first component necessary is a component to handle the recording and recognition of the user's speech. This is implemented using two Java classes. The primary class is Recognizer.

This class provides methods for beginning recording, stopping recording, performing recognition, getting back results from the recognizer, and comparing these results to the expected user utterances. The recording is performed using a helper class, Recorder.

When the *startRecording()* method is called, the Recognizer launches a new thread using the Recorder class, which finds the audio input device (the microphone) and begins to record. This thread continues to record until the *stopRecording()* method is called. The recording is done as a 16 kHz WAV file, and is saved to the file location given when the Recognizer is instantiated. The standard Java audio recording API is used for this process.

When the *doRecognize()* method is called, this audio file is processed by the speech recognition system. The speech recognition system is called as an external process. Therefore, Sphinx-2 must already be installed on the client machine. Sphinx-2 can be installed directly on UNIX-based machines, as well as under Cygwin on Windows machines. Full installation directions are available online at [9]. The installation involves the standard installation of Sphinx-2, plus the addition of certain shell scripts written to ease communication between these components and Sphinx-2. Thus, when the *doRecognize()* method is called, Sphinx-2 is started as a separate process and given arguments as to the location of the audio file and its language model files. The next sections describe where these language model files come from.

After the utterance has been recognized, either the *getBest()* or *getNBest()* methods are available to be called. These methods return the most likely utterance and the list of the N most likely utterances, respectively. These methods access the output of the last run of the recognizer to gather these utterances. The output returned by the *getBest()* method is similar to what one would get if using a dictation system. That is, Sphinx-2 returns a single utterance of what it believes is the most likely sentence spoken by the user, given the audio file and its language modeling files. The *getNBest()* method, on the other hand, can return a number of possible utterances. These utterances are ranked in terms of the recognizer's confidence in the fact that the student spoke the given utterance. An example of this will be shown at the end of this section on page 5.

The last method of Recognizer is the *getBestMatch()* method. This method takes as arguments a list of possible spoken utterances and a list of expected user responses. The list of possible spoken utterances is normally retrieved via the *getNBest()* method described above. The list of expected responses is defined by the exercise creator, as will be seen in the walk-through. This method compares these lists to come to a conclusion about what utterance, if any, the student has spoken. This method may need to be modified for different types of exercises, but in these utilities it works in the following way. For each of the possible spoken utterances received from the recognizer, the tool checks if it matches any of the expected responses. The first possible spoken utterance that matches an expected response is chosen. Note that this comparison is done by ignoring capitalization and punctuation, since Sphinx-2 does not include this information in its output. If none of the possible spoken utterances match any of the expected responses, the utterance "NotUnderstood" is returned. This indicates that it appears the user either said something different than the expected responses or the recognizer did not understand the utterance to be one of those expected responses. If the recognizer did not return any possible utterances, "NotHeard" is returned, which indicates that the recognizer was unable

to discern any words in the audio file. This usually indicates a problem with the audio hardware, such as the microphone or microphone settings. Thus, this method can return either "NotUnderstood", "NotHeard", or one of the utterances in the list of expected responses.

As an example, assume that the expected responses were the following:

- The girl ate the apple.
- The boy ate the pear.
- The bear ate the toy.

Also assume the list of N-best utterances from the recognizer was the following:

- the toy ate the pear
- the toy ate the bear
- the boy ate the pear

In this case, the *getBestMatch()* method would conclude that what the user said was "The boy ate the pear." This is because this utterance is the first utterance from the recognizer's results that matches one of the expected responses.

4.2 Tutor Integration Component

In addition to the recording and recognition component, something else is needed to integrate with the authoring tools. This integration can be done manually by the exercise designer, or it can be inherited by starting with an exercise template included in the tools. This template provides the following functionality: it implements event handlers for the start and stop recording buttons which handle all of the interactions with the recording and recognition component and the authoring tools.

The event handler for the start recording button initializes the Recognizer object with the location to save the audio file and calls the *startRecording()* method. The event handler for the stop button is somewhat more complicated. First, is calls the *stopRecording()* method of the Recognizer. Then, it calls the *doRecognize()* method. Next, it retrieves the list of the n best utterances from the recognizer with the *getNBest()* method. After that, it calls the *getBestMatch()* method with the list of N-best utterances and the designer-supplied list of the expected responses for this specific exercise (or portion of exercise). The utterance returned from this method call is either the utterance the student is believed to have spoken, or one of the error messages "NotUnderstood" or "NotHeard". Finally, the event handler sets the text of a textbox widget with this utterance and signals a widget update event. This update event causes the Cognitive Tutor Authoring Tools to notice that the text has been inserted into the textbox. Therefore, the authoring tools will react according to the user response, just as if the user had directly typed their response.

Additional example applications available online also show how to use interface widgets other than textboxes and how to handle multiple problems or parts of problems within one CTAT exercise[9].

5 Walk-Through of a Simple Example Exercise

In this section of the report, the process of building a simple speech-enabled tutor exercise will be explained. The exercise will be the first step of a dialog focused on politeness. The user will be presented with three sentences. One of these sentences will be incorrect because it is impolite. The other two will be correct, and would take the user down different paths of the dialog if the exercise continued beyond the first step. The user will be asked to choose one of the sentences, click record, speak the sentence, and click stop. The text of the user's choice will then appear in the textbox below. The user will receive feedback based upon the correctness of his or her choice.

The first step is creating the interface. Normally, an exercise designer can either start from scratch or from a template. A starting template for this problem is available online at [9], but this description will also include the elements already completed in the template. The first step is to create or open an existing Java GUI program in the GUI designer. Next, one must place the needed interface widgets on the window. If the template does not have a "Universal Tool Proxy" (UTP) widget on the interface, one can be dragged onto the form from the widget panel that came with the Cognitive Tutor Authoring Tools. All pseudo tutors need this widget. For this particular exercise, also needed are one JLabel for the directions and one JLabel for each of the options. These labels can be placed anywhere on the form and modified to hold the text of the three options. The three options used in the example exercise are the following: "I would like a table, please", "I would like a seat by the window", and "Give me a table." The first two will be designated as the correct responses, whereas the third will be considered incorrect.

For this exercise, buttons for starting and stopping recording are also needed. These are present in the template, but could also be manually added simply by dragging two JButton widgets onto the window. Note here that JButton widgets and not DorminButton widgets are being used. The former are normal buttons built into Java. The latter are special buttons that come from the Cognitive Tutor Authoring Tools. It is not necessary to use the authoring tools buttons here because one does not want the tutor to take any action when the user presses these buttons. These buttons are only used for one's own purpose. Finally, a textbox is needed for the user's response to be entered into automatically. For this, a DorminTextField is necessary, instead of a normal JTextField, because the tutor is expected to respond to changes in this widget.

If not using the template for this exercise, one must implement the event handlers for the buttons described in the section on the tutor integration component. The event handler code for the start button is shown below.

```
private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
    File audioFile = new File (audioFileLocation);
    File hypFile = new File (hypothesisFileLocation);
    recognizer = new Recognizer(audioFile, hypFile);
    recognizer.startRecording();
 }
```

The audioFileLocation and hypothesisFileLocation are Java Strings whose values depend on where the Sphinx-2 speech recognition system was installed. Notice that the code creates an instance of the Recognizer class and then calls the *startRecording()* method. The event handler code for the stop button is shown below.

```
private void stopButtonActionPerformed(java.awt.event.ActionEvent evt) {
    recognizer.stopRecording();
    recognizer.doRecognize();
    String s[] = recognizer.getNBest();

    String spoken = recognizer.getBestMatch(s, currentAnswerChoices);
    dorminTextField1.setText(spoken);

    FocusEvent fe = new FocusEvent(dorminTextField1, FocusEvent.FOCUS_LOST);
    dorminTextField1.focusLost(fe);
}
```

In this method, the *stopRecording()* and *doRecognize()* methods are called, followed by the call to the *getNBest()* method. Then, the call to *getBestMatch()* returns the utterance the system had determined the student to have spoken, which is stored in the String spoken. Finally, the text of the DorminTextField is set to this value, and a FOCUS_LOST event is passed to the text field. This triggers the authoring tools to handle the student input.

The last step of the interface creation process is to define the variable *currentAnswerChoices* used in the above call to the *getBestMatch()* method. This corresponds to the possible answer choices that the student is expected to speak. This can be defined in the TutorInterface constructor by the code below:

```
currentAnswerChoices = new Vector();
currentAnswerChoices.add("I would like a table, please.");
currentAnswerChoices.add("I would like a seat by the window.");
currentAnswerChoices.add("Give me a table.");
```

Although the interface is now complete, there is one more step that must be done before one can demonstrate student behavior and let students use the exercise. The speech recognition system must be provided with a language model and a pronunciation lexicon. These files can be automatically created using the online knowledge base tool available at [7]. The input to this tool is a sentence corpus file. To create a sentence corpus file, a text file must be created with one possible student utterance per line. In this exercise, the file would then consist of three lines:

I would like a table please
I would like a seat by the window
Give me a table

Note that these utterances are stripped of punctuation. Once this file has been uploaded to the online tool, the tool will provide the language model files and pronunciation lexicon needed. These files must be downloaded and placed in the Sphinx-2 current model directory.

Now that the interface has been completed and the speech recognition system has been provided the necessary configuration files, one can run the exercise and create the behavior recorder graph. The behavior recorder graph is begun by clicking on the "Author" menu at the top of the exercise window and then clicking "Create Start State". This creates the root of the behavior graph. Next, one creates a state for each of the possible values that can be inserted into the textbox. There is one state for each of the three answer choices, and one state each for "NotUnderstood" and "NotHeard". Recall that the message "NotUnderstood" indicates that the list of utterances returned from the recognizer could not be matched with any of the possible student utterances, and thus the student's response could not be understood. Recall also that the message "NotHeard" indicates that the recognizer could not detect any speech in the audio signal, which usually means there is a problem with the student's microphone. A state is created by inputting the answer choice or error message into the textbox. Note that although the student will complete this exercise by speaking, it is not necessary to speak to demonstrate student actions. One can simply type what the student is expected to speak. The new state will then appear in the behavior recorder diagram and become selected. To add another state for one of the other answer values, one must reselect the root of the behavior graph and then type another value into the textbox, which became blank again when the root was reselected. After states have been added for each of the options, the behavior graph can be annotated as described previously to mark certain paths, such as the one for "Give me a table", as incorrect, and to add feedback and hints along other paths. The resulting behavior recorder graph is shown in Diagram 2. The exercise is now ready to be used by students. Diagram 3 shows a student's correct response, and Diagram 4 shows an incorrect response.

Diagram 2

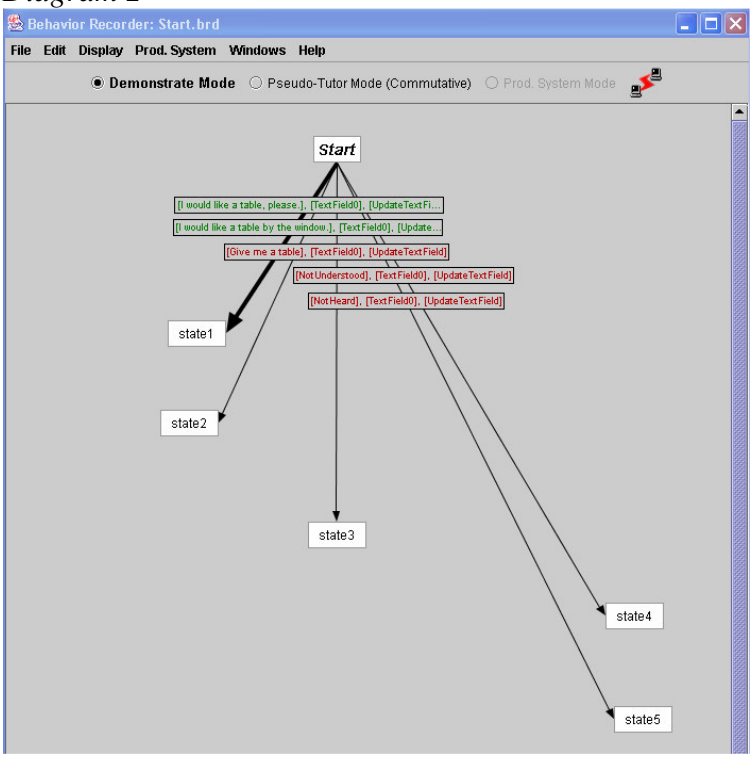


Diagram 3

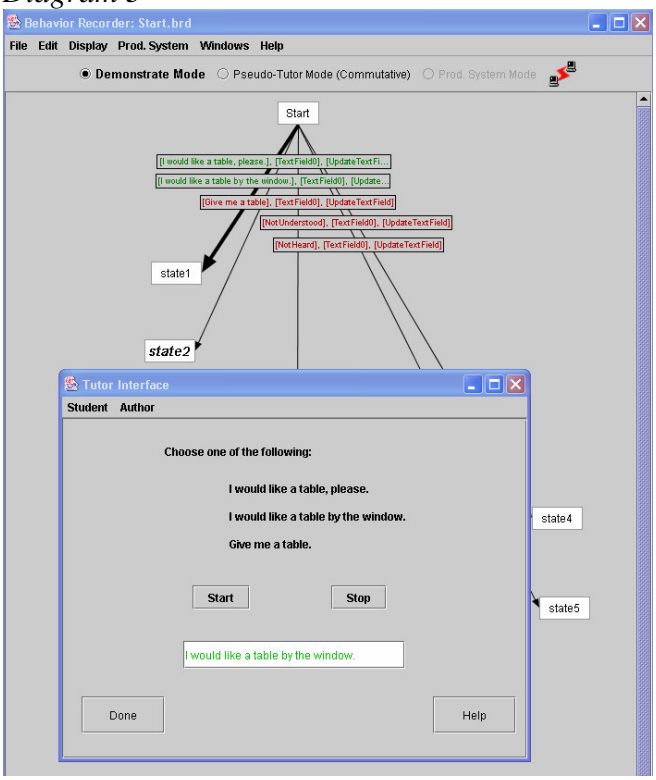
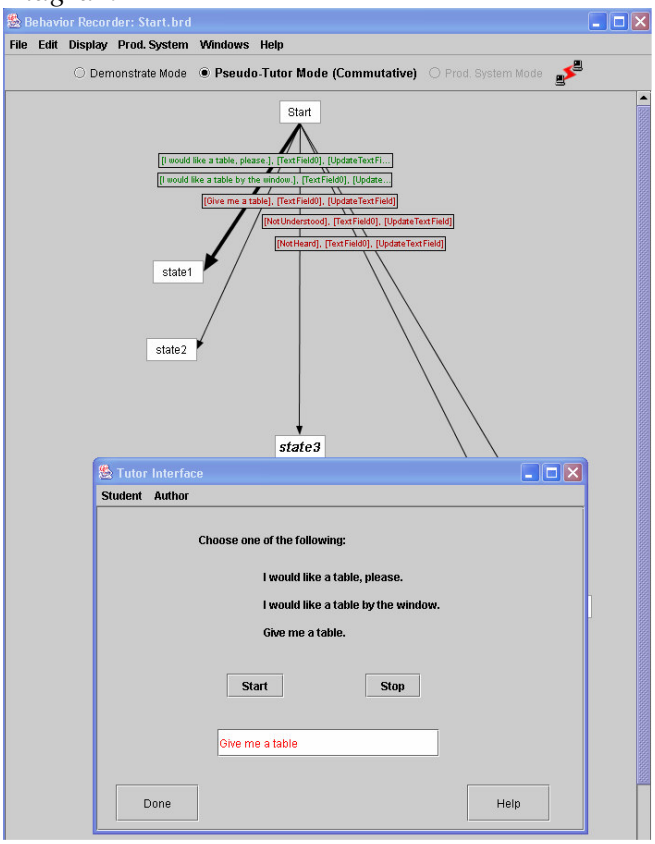


Diagram 4



6 Other Example Exercises

In addition to the simple example exercise given above, there are also two additional example exercises available online[9]. The first is named “Second Sight” and the second is named “You Are Here”. The first exercise requires the learner to speak sentences that are in the passive voice. This exercise uses a textbox for the user response, and automatically loads new problems when the user completes one. Diagram 5 shows a correct response, Diagram 6 shows feedback given for an incorrect response, and Diagram 7 shows feedback for the “NotHeard” message.

Diagram 5

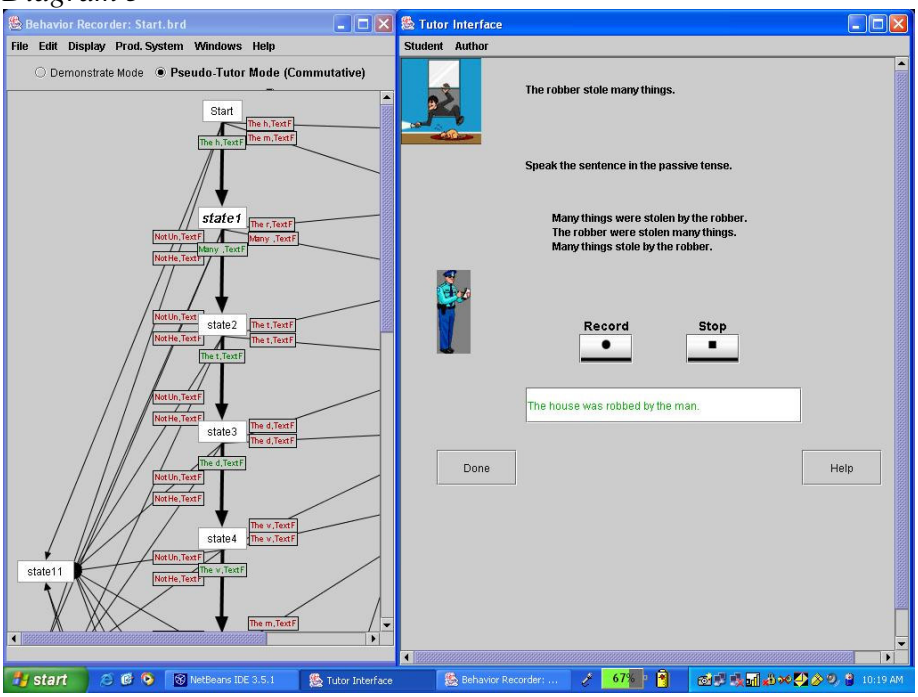


Diagram 6

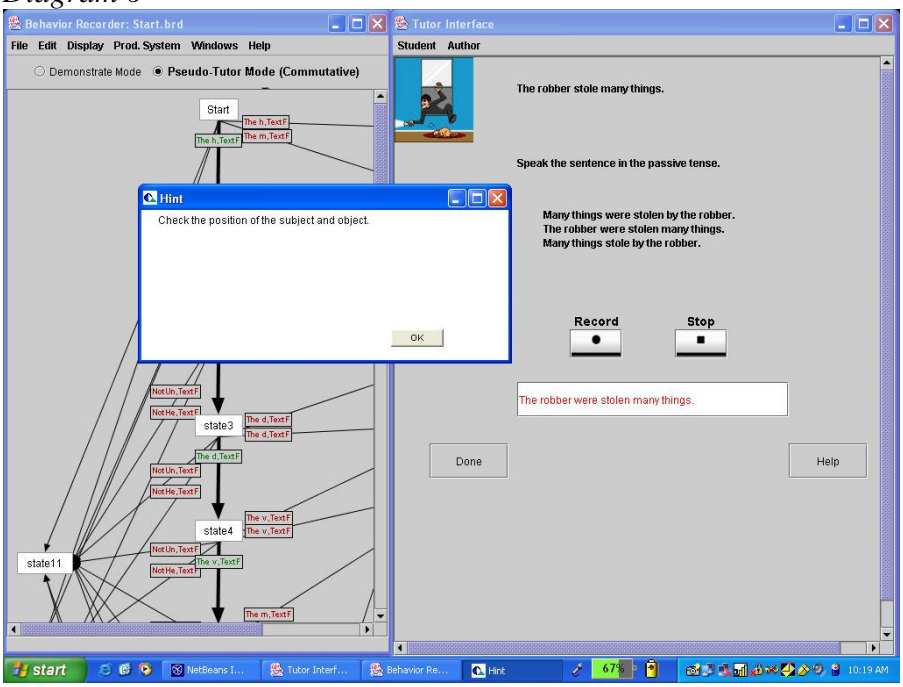
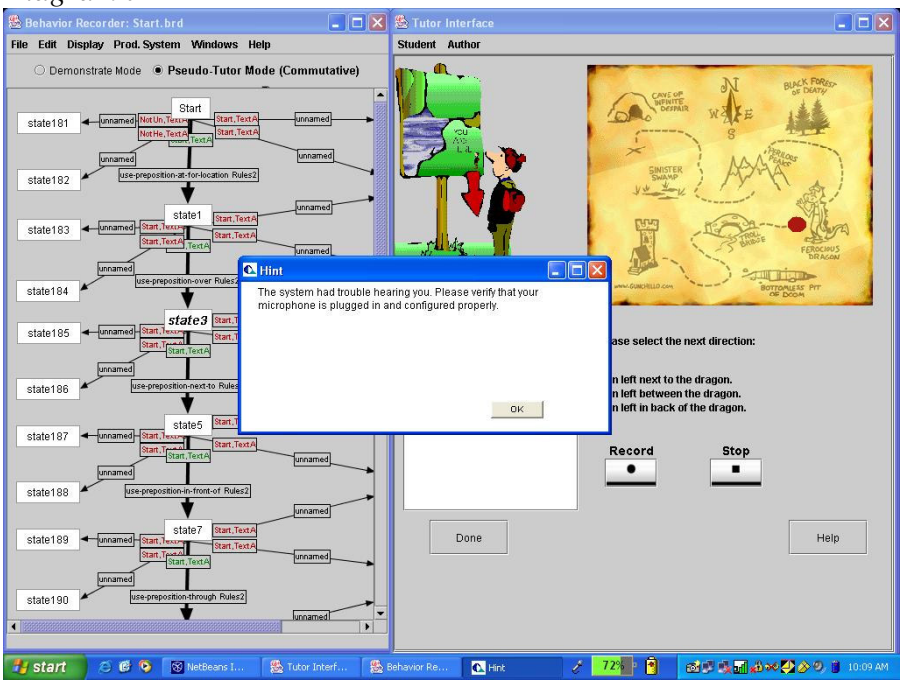


Diagram 7



The second exercise requires the user to navigate a hiker through a map, focusing on prepositions. It uses a new type of widget for user input, and also automatically progresses through subparts of the problem. Diagram 8 shows the interface after a series of correct responses, and Diagram 9 shows feedback for a hint request.

Diagram 8

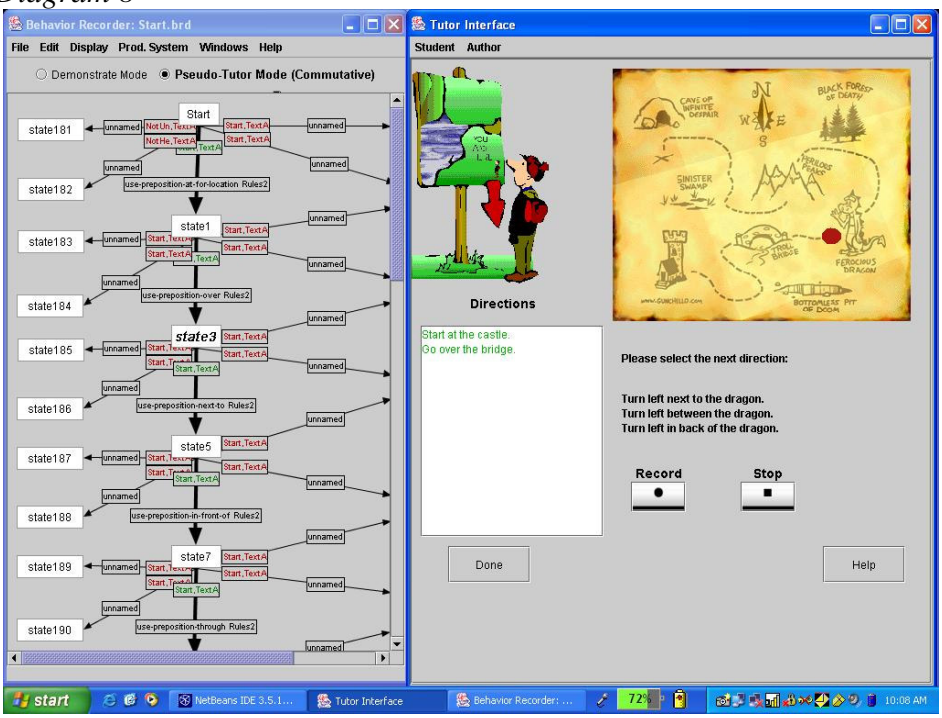
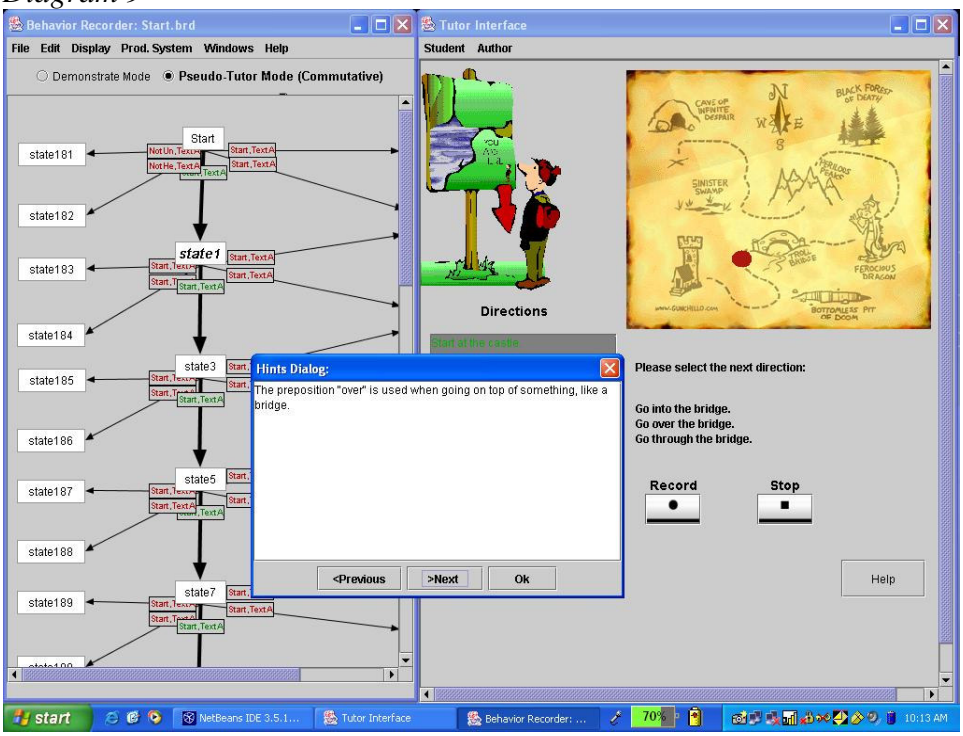


Diagram 9



All four diagrams also show examples of more complicated behavior recorder graphs than that of the simple example described.

7 Conclusion

The utilities described in this report were designed to make it simpler and easier to create speech-enabled tutors in the context of the Cognitive Tutor Authoring Tools. The reader has been walked through the creation of a simple speech-enabled tutor exercise and has been introduced to more complicated examples. The utilities described here are freely available online at [9].

8 References

[1] Koedinger, K. R., Aleven, V., & Heffernan, N. (2003). Toward a rapid development environment for Cognitive Tutors. In U. Hoppe, F. Verdejo, & J. Kay (Eds.), Artificial Intelligence in Education, Proc. of AI-ED 2003, 455-457.

[2] Koedinger, K., Aleven, V., Heffernan, N., McLaren, B. M., and Hockenberry, M. (2004). Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration. In the Proceedings of the Seventh International Conference on Intelligent Tutoring Systems (ITS-2004).

[3] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. The Journal of the Learning Sciences, 4 (2), 167-207.

[4] Corbett, A.T. & Anderson, J.R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. User modeling and user-adapted interaction, 4, 253-278.

[5] Huang, X., Alleva, F., Hon, H.-W., Hwang, M.-Y., Lee, K.-F. & Rosenfeld, R. (1992). The SPHINX-II speech recognition system: an overview, Computer Speech and Language, 7(2), 137-148.

[6] Sphinx Project Page: http://www.speech.cs.cmu.edu/sphinx/

[7] Sphinx Knowledge Base Tools: http://www.speech.cs.cmu.edu/tools/

[8] CMU Pronouncing Dictionary: http://www.speech.cs.cmu.edu/cgi-bin/cmudict/

[9] Sphinx2-CTAT Connection Utilities: http://www.cs.cmu.edu/~jonbrown/Sphinx2-CTAT/