Tools for Authoring Tutorial Dialogue Knowledge

Pamela W. Jordan and Carolyn Rosé and Kurt VanLehn
LRDC and Intelligent Systems Program
University of Pittsburgh
Pittsburgh PA 15260
pjordan@pitt.edu

Abstract. The current generation of intelligent tutoring systems (ITS) have successfully produced learning gains without the use of natural language technology, but the goal for the next generation is to add natural language dialogue capabilities. Since it is already a tremendous effort to add domain and pedagogical knowledge to the current generation of ITSs, adding natural language dialogue capabilities can further increase the development time by requiring that language knowledge also be engineered. Rather than having natural language knowledge become an additional engineering burden, we seek to build tools that will allow us to attack the problem of pedagogical and language knowledge engineering in tandem. In this paper, we describe the authoring tool suite we are building to address this problem. We have found that our prototype tools do facilitate the rapid development of natural language dialogue interfaces for ITSs. With these tools we were able to build knowledge sources for our dialogue interface to an ITS in only 3 man months. The resulting dialogue system was able to hold natural language dialogues with students on 50 physics concepts and students showed significant learning gains over seeing only monologue text hints [8].

1 Introduction

While the current generation of intelligent tutoring systems (ITS) have successfully produced learning gains [1, 4] without the use of natural language technology, the goal for the next generation is to add natural language dialogue capabilities. By doing so, ITS users will be able to type their answers and clarification questions in natural language in addition to using existing graphical user interfaces to communicate with the ITS.

Although it is already a tremendous effort to add domain and pedagogical knowledge to the current generation of ITS, adding natural language dialogue capabilities can further increase the development time by requiring that language knowledge also be engineered. Rather than having natural language knowledge become an additional engineering burden, we seek to build tools that will allow us to attack the problem of pedagogical and language knowledge engineering in tandem. In this paper, we describe the authoring tool suite we are building to address this problem. We have found that our prototype tools do facilitate the rapid development of natural language dialogue interfaces for intelligent tutoring systems. With these tools we were able to build knowledge sources for our dialogue system in only 3 man months. The resulting dialogue system was able to hold natural language dialogues with

students on 50 physics concepts and students showed significant learning gains over seeing only monologue text hints [8].

2 Background

Our tool suite was developed in the context of the Atlas project with the goal of assisting and semi-automating the authoring of all the domain specific knowledge sources required to apply the Atlas architecture in a specific tutoring system [3].

Currently, the Atlas architecture is being developed and integrated with the Andes physics tutoring system [11]. Atlas' task is to interact with students via natural language dialogue and encourage them to infer or construct target knowledge when Andes indicates they need this help during problem solving. This particular type of dialogue is called a knowledge construction dialogue (KCD). For example, Andes might simply tell the student that when an object moving in a straight line is slowing down, its acceleration is in the opposite direction to its velocity. Atlas will instead try to draw that knowledge out of the student with a dialogue like the one shown in Figure 1, where the student derives the target principle from a deeper one. KCDs are intended to provide deeper knowledge by connecting principles, relating them to common sense knowledge, and giving the student practice in talking about them.

1. T: Here's an example that you can use to remind yourself of the direction of a body's acceleration vector when it is slowing down. If a ball is rolling to the right and you want to stop it, which way do you have to push?
2. S: left
3. T: So then which way would the acceleration vector be pointing?
4. S: up
5. T: Since force is mass times acceleration, and mass is a scalar quantity, how would you expect the direction of the force vector and that of the associated acceleration vector to be related to one another?
6. S: same
7. T: If you push left, what direction is the associated force vector pointing?
8. S: left
9. T: So then which way is the acceleration vector pointing?
10. S: left
11. T: Whenever an object is slowing down, the net force acting on the object is opposite its direction of motion. Thus, when an object is slowing down, the direction of its acceleration is also opposite its direction of motion.

Figure 1: **Dialogue excerpt from the Atlas-Andes evaluation corpus**

Our basic approach for Atlas is to represent knowledge construction dialogues (KCDs) as networks of pushdown automaton (PDA). States correspond to tutor utterances (usually questions), arcs correspond to correct student responses, and pushes to incorrect student responses. Pushes call a subdialogue and pops return from one. Finite state networks are often used in spoken language dialogue systems (e.g. [7, 12] inter alia), so it makes sense to start with this basic approach and try to overcome their limitations.

Atlas, as shown in Figure 2, uses a robust parsing approach (CARMEL [10]) to understand the student's input and match it to expected inputs and a reactive planner (APE [2]) to manage

the dialogue by choosing and expressing the states that correspond to the tutor's responses to students. However to improve upon the finite state approach, we consider more than the expected arcs for the current tutor state so that if the student answers questions that we would have asked later in the current path, it will not ask it again but may just remind the student of what he had said at the appropriate time.

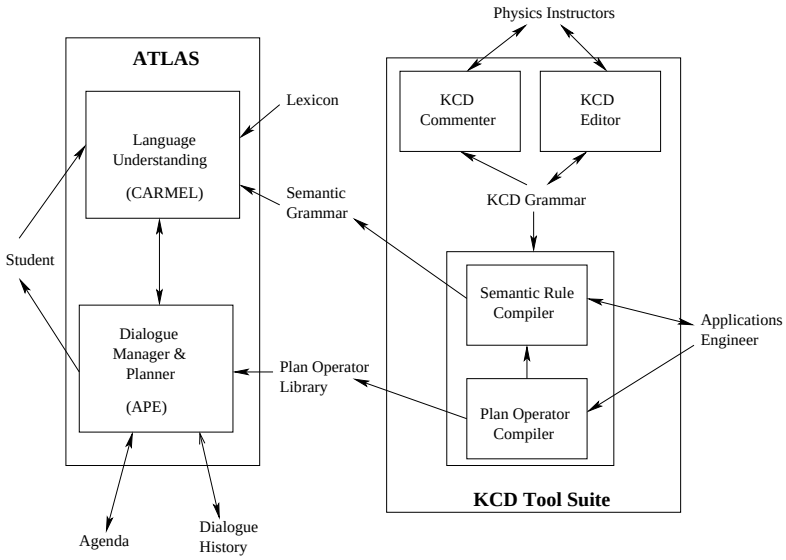

Figure 2: Atlas and the KCD Tool Suite

So far, Atlas is focusing on KCDs that teach domain principles for a small portion of physics and covers 50 principles. The prospect of building knowledge construction dialogues to cover this many principles was daunting enough to motivate us to expand our plans for building knowledge acquisition tools.

3 The tool suite overview

As shown in Figure 2, there are four tools that currently comprise the KCD authoring tool suite: (1) an editor 2) a commenter 3) a plan operator compiler and 4) a semantic rule compiler. Each of the tools uses a common KCD grammar which represents the target PDA. The two main outputs of the current prototype version of the tool suite are operators for the APE tutorial dialogue planner [2] and a semantic grammar for the CARMEL natural language understander [9].

The editor provides a graphical user interface that allows the author to enter the dialogue grammar by entering only the natural language strings that represent expected student dialogue contributions and how the tutor would respond to each of the student contributions. The author must also relate the strings to one another in appropriate ways as will be shown in Section 5. The editor then translates the author's inputs into the KCD grammar.

The commenter allows other project members and other instructors to review and comment on the content of the KCDs in order for the domain instructor to get feedback on explanations and questions that are not clear and to get ideas for alternative ways of explaining the same domain concepts. The commenter interprets the KCD grammar and presents the

appropriate content strings allowing the reviewer to focus on the domain knowledge instead of the underlying specification language.

The plan operator compiler translates the KCD grammar into domain specific reactive planning operators that indicate how to move from one tutor state to another in response to a classification of the student input. These operators also make use of the tutorial intentions included in the KCD grammar by recording them in the agenda as they need to be satisfied and in the dialogue history as they are satisfied. This record keeping allows the reactive planner to skip around in the network so that all the transition arcs do not need to be explicitly stated by the KCD author.

The semantic rule compiler extracts the expected student inputs from the KCD grammar, helps the author cluster and label semantically similar inputs and then generates semantic grammar rules that will map student inputs into the answer classes used within the plan operators. When the student types in a response to a tutor question, it gets parsed according to this semantic grammar and if the input is recognized by the grammar a semantic label is assigned to it. This semantic label allows answer classes within the PDA to be identified. In this way, Atlas can recognize expected responses that students may not have expressed with the same words and syntax as the KCD author.

4 The KCD grammar

```
<Recipe> ::= <intent><steps>
<steps> ::= [<Recipe>|<goal>]+
<goal> ::= <intent><tutor-contribution>[<possible-answers>]*
<possible-answers> ::= [<part-of-answer>]+
<part-of-answer> ::= <right-part>[<wrong-part>]*
<right-part> ::= <student-contribution> EXPECTED
<wrong-part> ::= <student-contribution>|$anything else$ WRONG <possible-
remediations>
<possible-remediations> ::= [<Recipe>|<goal>]+
<intent> ::= <string describing a tutor intention>
<student-contribution> ::= "<natural language string>"
<tutor-contribution> ::= "<natural language string>"
```

Figure 3: **Specification of the KCD Grammar**

Figure 3 shows the specification language for the KCD grammar. Currently, the grammar defines primitive actions and recipes. A primitive is defined to be a tutoring goal that is decompositionally a leaf node in a plan structure and an associated natural language string that realizes that primitive tutoring goal. A primitive may encode a tutor explanation or a question for eliciting a particular piece of information or both.

Recipes are higher level tutoring goals that are defined as a sequence of any combination of primitives and recipes [13]. This representational approach is widely used in computational linguistics since problem-solving dialogues and text are believed to be hierarchical structured and to reflect the problem-solving structure [5]. Tutorial intentions or goals are associated with both recipes and primitives. In this way, the author may encode alternative ways of achieving the same tutorial intention.

For each primitive tutoring goal, the grammar also includes information on what to expect from the student so that information on how to respond appropriately to the student can also be included in the grammar. Possible student responses are categorized as expected correct answers and a set of expected typical wrong answers. Student responses can be represented as parts when the answers are complex and it is possible for the student to give a partially correct answer. The reason expected student responses are broken into parts is to avoid having to list every possible combination of right and wrong answers. For example, with the tutor question "So what two factors influence the acceleration of an object moving along a circular path?" there are two parts to an expected right answer: "the speed of the object and the radius of the circular path". These two parts may appear in any order without affecting the correctness of the answer. Each part is independent of the others in that a student may get one part correct while missing the other. If the student only knows one factor in the above example or is wrong about one, then only that part gets remediated while if the student is wrong about both parts then both can be remediated.

For each answer part, we list a number of expected, typical wrong answers. For example, the student may have said the size of the object was a factor in the above example. For completeness, we always include $anything else$ in case part of the answer was left out of what the student said altogether. Every wrong answer has associated with it a list of tutorial goals that must be achieved by the tutorial planner in order to remediate that wrong answer.

5 The KCD editor

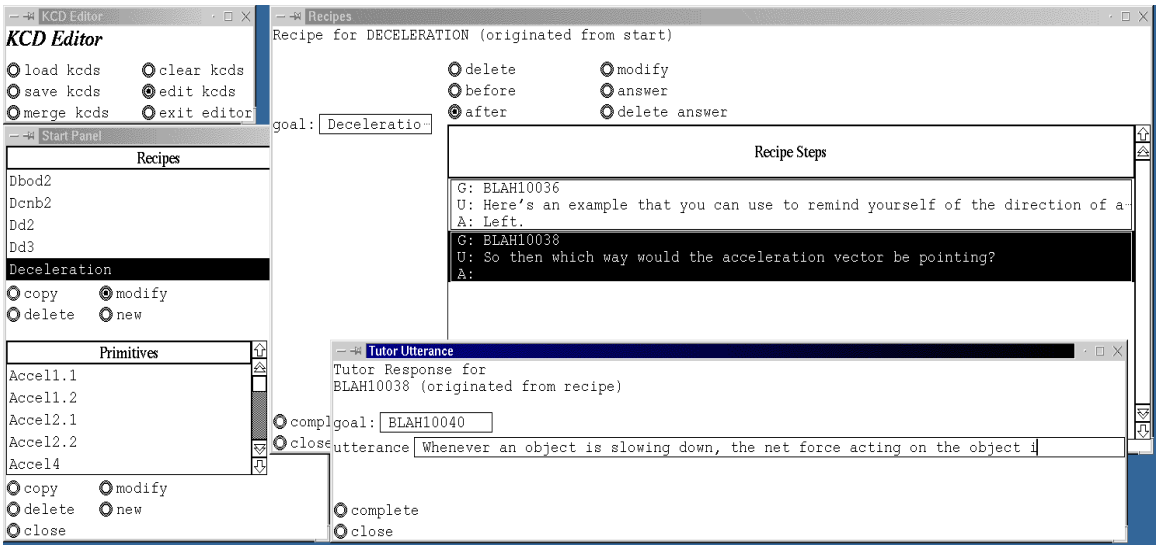

Figure 4: Snapshot of an editing session

Our primary tool is the knowledge construction dialogue (KCD) editor. Figure 4 is a snapshot of an editing session at the point of having inserted a new step. As can be seen here, the editor displays independent windows (4 in this example) depending on the actions taken within a currently active window. The author is free to move these windows around on his desktop and can minimize or close these windows as is convenient for him. Because the author can infinitely descend into remediations, the desktop can become cluttered and confusing. Until we can find a better presentation metaphor, we opted to allow the author to

manage the desktop by displaying conceptually similar parts of the KCD grammar in independent windows. We also include a textual trace in the top of each window to indicate what window and action in that window caused the current window to appear unless the window is a temporary pop-up window. This trace information is provided to aid the author in his desktop management task.

To explain the editing process in more detail, we will trace through part of the editing process. The KCD Editor window as seen in the upper left corner of Figure 4 is always visible and initiates and terminates editing sessions and allows the user to save the results of a session and reload it later for additional editing.

5.1 Authoring Recipes and Primitives

In the lower left window of Figure 4, the author chose to modify the KCD for deceleration which causes an overview of the main line of reasoning represented by the recipe to appear in the upper right window. The recipe shows the sequence of primitives and recipes that help realize the higher level recipe goal. Selecting a primitive step with the mouse and moving the cursor along the string allows the author read the entire natural language string if necessary. New steps can be inserted into a recipe with the *before* and *after* buttons and old steps removed with the *delete* button. The natural language strings associated with primitive actions, can be modified or entered with the *modify* button, and maintenance and authoring of expected student responses can be initiated for steps that are primitives with the *answer* button.

The Primitive overview window is similar to the Recipe window but allows the entry or modification of one goal and the tutor's natural language string to realize that goal. Maintenance and authoring of expected student responses (as described in the next Section) are also supported for primitive goals.

Returning to the editing of the deceleration KCD in Figure 4, the author next chose to insert two new steps *after* the first tutor-student interaction. Here we see the author entering in the tutor's natural language string for the second new step in the lower right window. We also see that there is a tutor utterance entered for the first new step but no expected student response.

After the author completes the entry of the tutor's string for the second new step, he will next go back and select the first tutor-student interaction and then the *answer* button in the upper right window. Doing so allows him to enter the expected student responses for this tutor string.

5.2 Authoring Anticipated Answers

Selecting the *answer* button in Figure 4 causes the Expected Answers window, shown in Figure 5, to appear. This window allows the author to create or modify the parts of an expected correct student answer to the tutor's utterance. The natural language strings that represent expected parts of a student's answer can be deleted or modified from this window and additional answer parts can be added with the *add &*. The *add |* button allows the KCD author to add in some alternative phrasings for the answer part. The addition of alternative phrasings is not mandatory at this point in creating KCDs but it is helpful for the semantic grammar compilation process described in Section 8. Semantically different correct answers must be added as

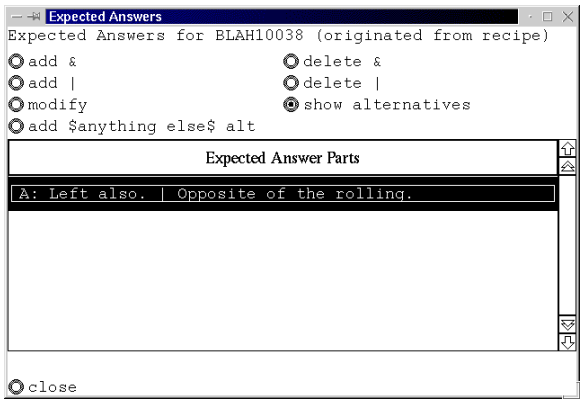

Figure 5: Expected Answers window

additional answers by returning to the recipe window in Figure 4 and selecting *answer* again for the same step in the recipe.

Associated with each correct answer part are sets of expected wrong answers (alternative answers) and maintenance and authoring of these expected wrong answers can be initiated from this window with the *show alternatives* button.

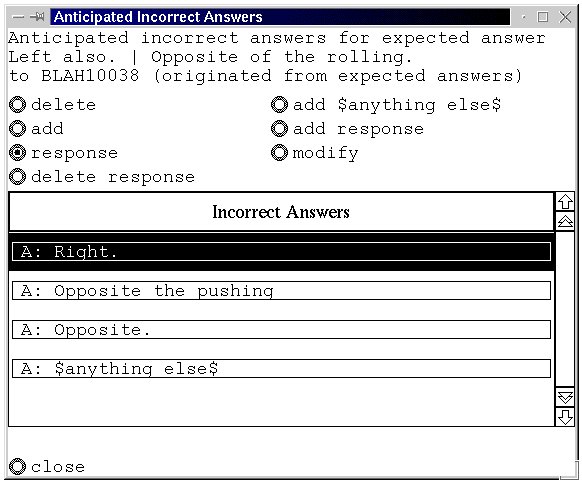

Figure 6: Anticipated Incorrect Answers window

Selecting the *show alternatives* button in Figure 5 causes the Anticipated Incorrect Answers window in Figure 6 to appear. This window allows the author to modify or enter additional wrong answers that he might wish to try to remediate. Since we are only dealing with natural language strings at this point, if the remediation or response would be the same for two very different answers then the KCD grammar author will need to specify both strings separately but could assign the same remediation to both. A future enhancement will allow the author to group together wrong answers that he would remediate in the same way.

Currently, the only artificial language string we allow is $anything else$. This is a default case for when the student answer was not anticipated or is unrecognized by the natural language understanding component of the dialogue system. However, it does allow the author to do a very limited type of grouping of student answers so that he can assign them all the same remediation.

5.3 Authoring Remediations

Finally for each anticipated incorrect answer the author can initiate a remediation KCD (a push to another KCD) from the window in Figure 6. Choosing the *response* button allows the author to enter one recipe or primitive to remediate this wrong answer. If no remediation was previously defined then the response panel window in Figure 7 appears but if one is already defined then a recipe window or primitive window that summarizes the remediation appears.

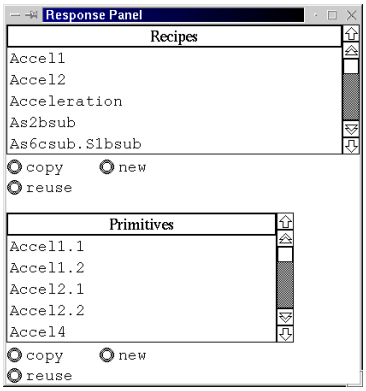

Figure 7: Response Panel window

The Response Panel window in Figure 7 allows the KCD author to choose an existing primitive or recipe for a remediation or to create new ones. If a new recipe or primitive is needed for remediation then the author selects the *new* button within the proper bank to bring up the appropriate blank recipe or primitive summary window (e.g. either a recipe or primitive must be anticipated beforehand).

6 The KCD commenter

In a limited sense, the KCDs are intended to be better than naturally occurring dialogues. Just as most text expresses its ideas more clearly than informal oral expositions, the KCD is intended to express its ideas more clearly than the oral tutorial dialogues that human tutors generate. Thus, we need a way for expert physicists, tutors and educators to critique the KCDs and suggest improvements. Since the underlying dialogue grammar can be complex, it is not useful to merely print it out and let experts pencil in comments. Our second tool facilitates this by allowing expert physicists and psychologists to navigate around the network and enter comments on individual states (see Figure 8). It presents a dialogue in the left column, and allows the user to enter comments in the right column. Since there are many expected responses per tutorial contribution, the user can select a response from a pull down menu. This causes the whole dialogue to adjust, opening up new boxes for the user's comments. This tool runs in a web browser, so experts can use it remotely.

7 The plan operator compiler

From the KCD grammar we can generate plan operators that are interpreted by a reactive planning engine [2]. The engine does not simply follow the PDA. Instead, it has rudimentary (but growing) capabilities for treating the network as a plan for the conversation that it will

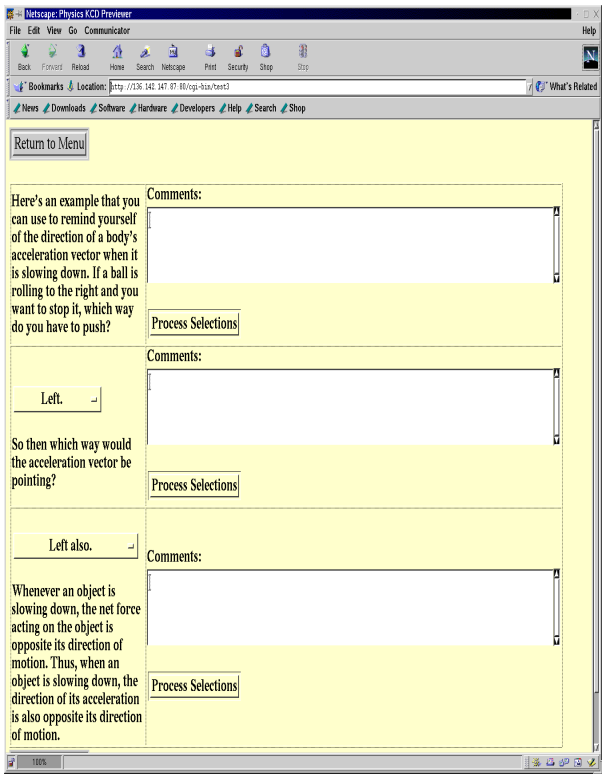

Figure 8: The KCD commenter

adapt as necessary. For instance, in the conversation of Figure 9, suppose the student said at line 2, “The acceleration makes the velocity vector longer, so the elevator should be going faster.” The reactive planner should recognize that the student has skipped ahead in the conversation plan, so it should have Atlas say line 7 instead of line 3.

As mentioned earlier the plan operators use the tutorial intentions in the dialogue grammar specification to keep track of what needs to be satisfied and what has recently been satisfied. This is part of what allows the reactive planner to skip around in the network when necessary. As the above example shows, this allows for increased initiative on the part of the student. In addition, the plan operator language allows for failure points. These could be due to either a KCD failing to have the desired effect on a student or a misunderstanding of an explanation or question. At a failure point, APE can back up and try again to satisfy an intention by trying a different path. However, we currently have this feature turned off in the operator compiler because we first must introduce more flexibility in generating the surface form of the tutor’s responses.

Currently the plan operators map directly to strings but in our next version of the plan operator compiler we plan to map to natural language templates and eventually to propositions. We are in the process of investigating the possibility of merging the template authoring tool, TATTOO, which is part of the YAG architecture for dialogue generation [6], with the KCD editor. Whether we succeed or not at the tool merger, we are independently considering using YAG for natural language surface form generation.

1. T: As you know, you can think of an acceleration vector as changing a velocity vector by pulling on its tip. In this case, the elevators velocity vector points straight down and you drew its acceleration vector pointing down too. What does your acceleration vector do to the velocity vector?
2. S: It stretches it downward.
3. T: Does that make it longer?
4. S: Yes.
5. T: Which means the elevator is speeding up, right?
6. S: Yeah, but its not really speeding up. Its slowing down.
7. T: Right. So maybe your acceleration vectors direction is wrong. What direction should it be?
8. S: Up.
9. T: Right. So what have you learned about the direction of acceleration when an object is moving in a straight line and slowing down?
10. S: Its in the opposite direction from the velocity.
11. T: Yes! Thats exactly right. Good job.

Figure 9: **A hypothetical dialogue between Atlas and a student**

8 The semantic rule compiler

Although the authors only see and edit natural language text, we cannot expect students to type in exactly the responses that the authors enter. To handle this variability, we translate the expected student responses into more general semantic representations that factor out inconsequential syntactic and lexical variations. To extract this general semantic knowledge from the expected student input strings created during KCD authoring and relate it to the syntactic and lexical variations, we use the semantic rule compiler. It helps the applications engineer rapidly develop a domain specific sentence level language understanding component in a two phase process that requires guidance and feedback from the applications engineer.

During the first stage, the semantic rule compiler presents the engineer with clusters of input strings from the KCD grammar. The clustering is based on the degree to which strings share content words. The engineer can pick and choose which strings belong in the cluster, can add more strings that he composes himself or he can compose a query so that the compiler can find other relevant input strings in the KCD grammar specification.[1] The engineer then associates conceptual labels with each cluster. For example, the engineer might cluster the two strings “it changes direction” and “change the direction” and assign it a concept label of <concept1>. During the second stage, the semantic rule compiler takes the strings and conceptual labels as input and induces semantic patterns for determining which expected input the student’s input best matches. The learned patterns currently take the form of string templates. For example for the two strings above the semantic rule would be *<concept1>* $\longleftrightarrow$ *change direction*. Finally the compiler associates an answer category to the concept that the plan operators recognize.

So when the student types a response to a tutor question, CARMEL (our natural language understanding module) stems each of the words and then parses the stemmed input using the semantic grammar. The parser skips words as necessary to produce a parse but prefers

[1] This is similar to Key Word in Context (KWIC) tools.

the parse with the least amount of skipping. Once a student input string is accepted by the semantic rules, the appropriate concept label is assigned to the parse as well as the answer category recognized by the plan operators.

We plan to improve the semantic rule compiler in the future so that it also takes CARMEL's syntactic analyses as input and generates constructor functions that will attach to CARMEL's lexicon. These semantic constructor functions will map CARMEL's syntactic representation to a semantic representation. A further improvement we have planned occurs at the end of the second stage of the compilation. The future compiler will also generate new strings for the conceptual labels it was given as input and the engineer can provide feedback to the learning process in order to improve the patterns it learns. The future version of the semantic rule compiler is described in detail in [9].

Although the semantic rule compiler is still very simple compared to the future version we have planned, it took us only three days to create a semantic grammar for CARMEL that supports 55 KCDs.

9 Conclusions and Current Status

Our main objective in developing our authoring tool suite was to allow instructors and developers with little or no computational linguistics expertise to rapidly develop dialogue interfaces for their areas of instruction. Thus, the authoring interface was designed to insulate the author from the underlying computational linguistics aspects of dialogue planning and language understanding. The author is free to focus fully on the most effective way for teaching particular concepts, how misconceptions and missing knowledge manifest themselves in student answers, and which aspects of student answers are relevant for evaluating their content.

We are willing to distribute the components of the tool suite to others in the community who would be interested in using it but with the caution that it is still being improved and many of the components are not yet portable to other computing environments (e.g. we develop our software under the Linux operating system instead of Microsoft Windows). The tool suite is still a work in progress because we are not yet using the full powers of APE, the dialogue planner and manager, and CARMEL, the natural language understander.

Approximately four people have used the commenter and it appears that it does not require special training to use. We have two KCD authors presently using the KCD editor. It takes about one week of full-time use to become comfortable with the editor; what is more difficult and time consuming is figuring out what to say in the KCDs. The plan operator compiler does not require intervention and so does not require special training whereas the semantic rule compiler does. So far only one project member has used the two compilers.

The prototype tool suite was recently used to develop knowledge sources for implementing directed lines of reasoning targeting 50 physics principles covering all aspects of Newtonian mechanics. The entire authoring process, including initial authoring, review by collaborating physicists, revision, and compilations of knowledge sources required only three man months of development time. The result was a running dialogue system that was pilot tested with both people who have previously studied physics and with students currently enrolled in freshman physics courses.

References

[1] A. Corbett, John Anderson, Arthur Graesser, Ken Koedinger, and Kurt VanLehn. Third generation computer tutors: Learn from or ignore human tutors? In *Proceedings of the 1999 Conference of Computer-Human Interaction*, pages 85–86, New York, 1999. ACM Press.

[2] Reva Freedman. Plan-based dialogue management in a physics tutor. In *Proceedings of the 6th Applied Natural Language Processing Conference*, 2000.

[3] Reva Freedman, Carolyn Rosé, Michael Ringenberg, and Kurt VanLehn. ITS tools for natural language dialogue: A domain-independent parser and planner. In *Proceedings of the Intelligent Tutoring Systems Conference*, 2000.

[4] Abigail Gertner and Kurt VanLehn. Andes: A coached problem solving environment for physics. In *Proceedings of the Intelligent Tutoring Systems Conference*, 2000.

[5] Barbara Grosz and Candace Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12:175 – 204, 1986.

[6] Susan McRoy, Songsak Channarukul, and Syed S. Ali. Text realization for dialog. Technical Report FS-00-01, Papers from the AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications, 2000.

[7] Norbert Reithinger and Elisabeth Maier. Utilizing statistical dialogue act processing in Verbmobil. In *ACL95, Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 1995.

[8] Carolyn Rosé, Pamela Jordan, Michael Ringenberg, Stephanie Siler, Kurt VanLehn, and Anders Weinstein. Interactive conceptual tutoring in Atlas-Andes. In *Proceedings of AI in Education 2001 Conference (in press)*, 2001.

[9] Carolyn P. Rosé. Facilitating the rapid development of language understanding interfaces for tutoring systems. Technical Report FS-00-01, Papers from the AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications, 2000.

[10] Carolyn P. Rosé. A framework for robust semantic interpretation. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000.

[11] Kurt VanLehn, Reva Freedman, Pamela Jordan, Chas Murray, C. Oran, Michael Ringenberg, Carolyn Rosé, K. Schultze, Robert Shelby, D. Treacy, Anders Weinstein, and Mary Wintersgill. Fading and deepening: The next steps for ANDES and other model-tracing tutors. In *Proceedings of the Intelligent Tutoring Systems Conference*, 2000.

[12] Marilyn Walker, Jeanne Former, and Shrikanth Narayanan. Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *Proceedings of COLING-ACL'98*, 1998.

[13] R. Michael Young and Johanna D. Moore. DPOCL: A principled approach to discourse planning. In *Seventh International Workshop on Natural Language Generation*, pages 13–20, Kennebunkport, Maine, 1994.

10 Acknowledgments

This research was supported by the Office of Naval Research, Cognitive Sciences Division under Grant No. N00014-00-1-0600 and by the National Science Foundation under Grant No. 9720359 to the Center for Interdisciplinary Research on Constructive Learning Environments at the University of Pittsburgh and Carnegie-Mellon University.