

A Scalable, Reusable, Spoken Conversational Tutor: SCoT¹

Karl SCHULTZ, Elizabeth Owen BRATT, Brady CLARK, Stanley PETERS, Heather PON-BARRY, Pucktada TREERATPITUK

*Center for the Study of Language and Information—Stanford University
220 Panama St., Stanford, CA 94305*

Abstract. Scalability and reusability of tutorial dialogue systems is a function of the corresponding characteristics in their component tutoring and dialogue technologies. This paper discusses an architecture for a scalable, reusable, spoken conversational tutor, SCoT. With this design we hope to minimize the efforts needed to reuse the components for implementing a tutor in any domain, large or small.

Introduction

This paper discusses an architecture for a scalable, reusable, spoken conversational tutor (SCoT), and one specific instantiation of it—SCoT-DC, which engages students in a spoken discussion of their solutions to problems of controlling damage aboard a ship. Although the subject matter knowledge that a tutoring system utilizes can usually not be employed in teaching a different subject, many tutorial techniques are largely reusable across subject domains. From the beginning of the system's design, an important goal was to achieve significant independence from subject matter and from tutoring-specific aspects of spoken dialogue. A dialogue system should separate conversational intelligence (i.e., how to use dialogue mechanisms in conversation) from the activity that the dialogue wants to accomplish (i.e., tutoring a student). In this respect we have adopted Clark's joint activity theory of dialogue [1]. We describe here the approaches taken to achieve this independence and separation, and assess the ways and the extent to which the effort has succeeded thus far.

Broadly speaking, our approach has been to abstract out tasks such as detecting whether a tutor's interlocutor possesses a given piece of knowledge, is ignorant of it, or holds a distorted form of that knowledge. We also abstract methods a tutor could employ for addressing gaps or confusions in knowledge, and for reinforcing a student's acquisition of correct knowledge. Likewise, we abstract strategies for deploying these tools in a tutoring session, and tactics for reacting to a student's immediate needs during a session. Dialogue systems also utilize linguistic and communicative knowledge, which may be symbolically represented in a fashion that is relatively independent of any particular use and domain. These abstractions and their uses in tutoring are formally modelled as a joint activity engaged in by tutor and student.

The rest of this paper is organized as follows. Section 1 describes a general purpose dialogue manager which handles multi-modal, mixed initiative dialogue. Section 2 describes in greater detail how the tutor is designed and how it makes use of subject matter knowledge

¹ This work is supported by the Department of the Navy under research grant N000140010660, a multi-disciplinary university research initiative on natural language interaction with intelligent tutoring systems. We wish to express our gratitude to our two anonymous reviewers. To see a demo or get more information please go to <http://www-csli.stanford.edu/semlab/muri/>.

and of spoken communication with a student. Section 3 describes how we facilitate natural language understanding and generation, both the techniques and technology used. Finally, we describe in Section 4 our framework for knowledge representation.

1. Dialogue Manager

Our Dialogue Manager (DM) is developed in the Architecture for Conversational Intelligence framework [2]. This architecture was originally used in the WITAS system, a *user*-driven, mixed-initiative dialogue system for controlling a robotic helicopter [3]. The generality of the framework in terms of handling dialogue moves in structured discourse has enabled us to adopt it for our *system*-driven, mixed-initiative tutoring dialogue system.

1.1 Conversational Intelligence

The DM is a domain independent component that handles the conversational intelligence of the system. It sits between the user and the domain specific Behavioral Agent (BA)—in this case the tutor—managing communication. The DM is responsible for interpreting any user input as a dialogue move or series of moves. It then determines the effects on the states of nodes in the Activity Tree (see Section 1.2), a hierarchical representation of the joint activities shared by the user and the BA. Also, the DM is responsible for turning a dialogue activity request by a BA into the appropriate dialogue moves directed towards the user.

By separating conversational intelligence from behavioral intelligence, we focus on developing the BA at the activity level. This allows the conversational intelligence to be independent of any domain specific features as well as being independent from the goal of the conversation. As such, the DM is intended to be reusable for any dialogue, including discourse other than tutoring or robot control. Not only does this mean that adapting the framework for a new domain would require less time and resources, but also that any future features that would be added for one domain could carry over to other domains as well. The current DM supports turn management, gesture input/output management, handling of syntactically or contextually uninterpretable user input, automatic insertion of discourse markers, appropriate pausing between utterances, and utterance aggregation.

In SCoT, the DM mainly handles the input/output flow while its subcomponents are responsible for specific features of dialogue. For instance the TurnManager subcomponent implements turn-taking algorithms and the SystemAgenda subcomponent handles utterance priority and utterance aggregation. By encapsulating these features into subcomponents, we can replace any features with domain specific versions without changing the DM core. This is most relevant for a subcomponent of the DM which needs to be domain specific—the subcomponent which helps the DM match user utterances to the appropriate dialogue context. This is very important for natural language understanding (see Section 3.1).

1.2 Activity Tree

The Activity Tree is the communication interface between the core DM and the domain-specific BA. It provides a way to structurally represent the joint activities of the student and the BA. Each node in the tree corresponds to an activity. Activities are categorized into two types: complex activity and atomic activity. An atomic activity corresponds to a single action supported by the system such as a question/answer pair, an explanation, a hint, or

GUI manipulations (gestures to or from the user). A complex activity, on the other hand, could contain other subactivities, both complex and atomic. The Activity Tree is manipulated in the following ways: 1) changing the state of some activities, 2) adding new activities, 3) deleting some activities, or 4) reordering activities. These manipulations are generic and independent of any single domain.

Each activity node is in one of the following defined states: planned, current, done, or cancelled. Each node can also have activity properties which contain domain specific information. This information is used mainly by the BA, but also for some context matching by a domain specific subcomponent in the DM. Additionally, either the DM or the BA can add a node to the Activity Tree to propose a new joint activity. The Activity Tree also provides the ability for both parties to navigate the conversation tree, such as reviewing old activities, or skipping certain activities.

In general, the DM only needs to understand the activity at an abstract level (e.g., the state of the activity) in order to perform the appropriate dialogue move, such as moving on to the next topic of the conversation. This means that the BA, regardless of the domain, only needs to provide high level information about its behavioral model, in order to take advantage of the Conversational Intelligence provided by the DM. This mainly implies that it is up to the BA, and not the DM, to decide the order of topics and to provide a hierarchy of sub-topics (so, for example, the DM can appropriately understand the distinction between top-level vs. low-level topics). The BA might also need to provide certain high-level domain-dependent context for the activity in order for the DM to carry out appropriate dialogue moves, but this domain specific information need not be complex and would only be used by a specialized DM subcomponent, with no effect on DM behavior for other domains. Section 2.3 shows in more detail what the Activity Tree looks like and how a BA would use it.

2. Tutor

2.1 Overview

In this section, we describe the components of our tutoring model that enable SCoT to develop global plans for reflective tutorial dialogue and respond dynamically to student spoken input. Crucially, the tutorial expertise modelled by each of these components is independent of domain knowledge (see Section 4) or conversational intelligence (see Section 1). For this reason, we suspect that the tutoring model will be reusable in other tutorial domains; e.g., mathematics. We begin by laying out some assumptions that have guided our work to date, and then turn to the details about our tutoring model.

In general, spoken language tutorial dialogue systems must be able to plan sequences of atomic actions (questions, answers, feedback), based on higher-level input. Inputs can include a problem, the student's solution, domain knowledge, an analysis of the student's spoken input (questions, answers, etc.), or a student model. On the basis of this input, the tutor carries out planning. For example, the tutor may need to develop an initial overall tutoring plan or a plan for responding to a student's incorrect answer.

SCoT's tutoring model covers domains (like damage control) that are non-deterministic (e.g., actions have unexpected outcomes) and have a dynamic problem state (e.g., a problem increases in complexity over time) [4], as well as domains (like mathematics) that are deterministic and static. The tutoring model can be divided into two components: strategies and tactics. We will briefly discuss both of these components and how they are realized in one particular deployment of this model, SCoT-DC.

2.2 Tutorial Strategies

The first component of SCoT's tutorial intelligence is *strategies*. Tutorial strategies are methods for constructing an initial plan for post-practice reflective dialogue. SCoT uses information in an annotated record of the student's performance in a problem-solving session (e.g., a series of physics problems, a session with a damage control simulator) to construct an initial overall tutoring plan; i.e., what problems are going to be discussed, what are the key points about them, what are the goals of discussion. In our current implementation, SCoT-DC, the tutor uses information from a record of the student's performance in a session with the damage control simulator DC-Train [5] to create an initial tutorial plan; i.e., what problems (e.g., shipboard crises such as fires) to review. Note that the initial overall tutoring plan can be dynamically revised during the tutorial dialogue (e.g., the student can request that the tutor skip discussion of a particular topic, the tutor can decide to drill the student on a particular knowledge point if a deficiency is shown).

SCoT presently has one main strategy for constructing the initial overall tutorial plan. Work by Katz et al. [6,7] has shown that reflective tutoring has a positive effect on learning and enhances the acquisition of strategic and conceptual knowledge. For these reasons, SCoT is designed to allow for reflective tutoring (i.e., to generate plans for post-practice reflection). The initial reflective tutorial plan is represented in the Activity Tree, and it has three main branches corresponding to an initial summary, topics for discussion, and a final summary.

In the initial summary, the tutor informs the student of how many problems arose, how many were resolved, and gives a brief appraisal of the student's performance. The tutor then selects a handful of exemplar problems—problems where the student's performance was especially poor. If more than one problem of a given type occurred, the tutor picks the one with the most errors. The motivation for this particular choice is that the student's knowledge and misconceptions will be reflected in the errors they make and that exemplar problems will make for the most interesting dialogues and create the most opportunities for learning. For each exemplar problem, the tutor introduces it, gestures to the student (e.g., highlighting a location on the ship), and with the student jointly reconstructs the observed (what the student did) and ideal (what an expert would do) methods of handling the problem by applying its tutorial tactics. Following the discussion of exemplar problems, the tutor may want to drill the student on one or more specific knowledge points (or give the student an option to do so). In the final summary, the tutor comments on the student's strengths and weaknesses, then reiterates the most important lessons from the tutorial dialogue.

This strategy has been primarily used as a proof-of-concept. We envision the tutor possessing a number of strategies to choose from, based on a student's level of knowledge and/or personal preferences (e.g., only stating incorrect student actions, discussing knowledge points instead of specific problems). We believe that this strategy, and other similar strategies, will work for a variety of domains for post-practice reflection. Knowing which strategies will produce the best results is something we hope to understand better through system evaluation, since domain-specific and individual user attributes influence the choice of one strategy over another in human-to-human tutorial interactions.

2.3 Tutorial Tactics

The second component of SCoT's tutorial intelligence is a repertoire of tutoring *tactics* (e.g., hinting). Student input, typically in response to a question, will initiate a tutoring tactic. The tutor searches the library of tutoring tactics to find all of the tactics whose

preconditions are satisfied in the current context. Like the plan operators in other systems [8], each tutorial tactic has a goal, a set of preconditions and a multi-step recipe [9]. Preconditions are currently based on two parameters: the classification of the student's answer in the tutorial dialogue and the classification of the student's action in the problem-solving session. In SCoT-DC, both classifications are created by comparing a student action/answer to a known 'correct' response that was generated by an automated expert agent within DC-Train (the damage control simulator) [5]. Recipes are composed of a sequence of actions. Actions in a recipe specify the tutor's responses and make updates to the Activity Tree (see Section 1.2). They can be primitive actions like providing feedback or complex actions like an embedded tutoring tactic.

SCoT's tactics are not specific to any one domain. Rather, dialogue with SCoT roughly matches the 5-step dialogue frame that Graesser and Person [10] observed in human-to-human tutoring, given directly below. Step 4 consists of familiar remediation techniques like hinting and further prompts to elicit the correct answer.

- Step 1: Tutor asks question (or presents problem)
- Step 2: Learner answers question (or begins to solve problem)
- Step 3: Tutor gives short immediate feedback on the quality of the answer
- Step 4: The tutor and learner collaboratively improve the quality of the answer
- Step 5: The tutor assesses the learner's understanding of the answer

The classification of student actions in a problem solving session currently can be one of the following: correct, incorrect, suboptimal, an error of omission, or an error of commission. The classification of student answers in a dialogue can be one of: correct, incorrect, partially correct, identification of another step in solving the problem, or the student admits to not knowing. Each combination of student answer and student action classification has a different tutorial tactic. The following tactic from SCoT is for responding to a partially correct answer, where the student's corresponding action in the problem-solving session was suboptimal:

Goal: Discuss Partially Correct Answer/Suboptimal Action

Preconditions

- Student Answer = Partially Correct
- Student Action = Error of Omission

Recipe

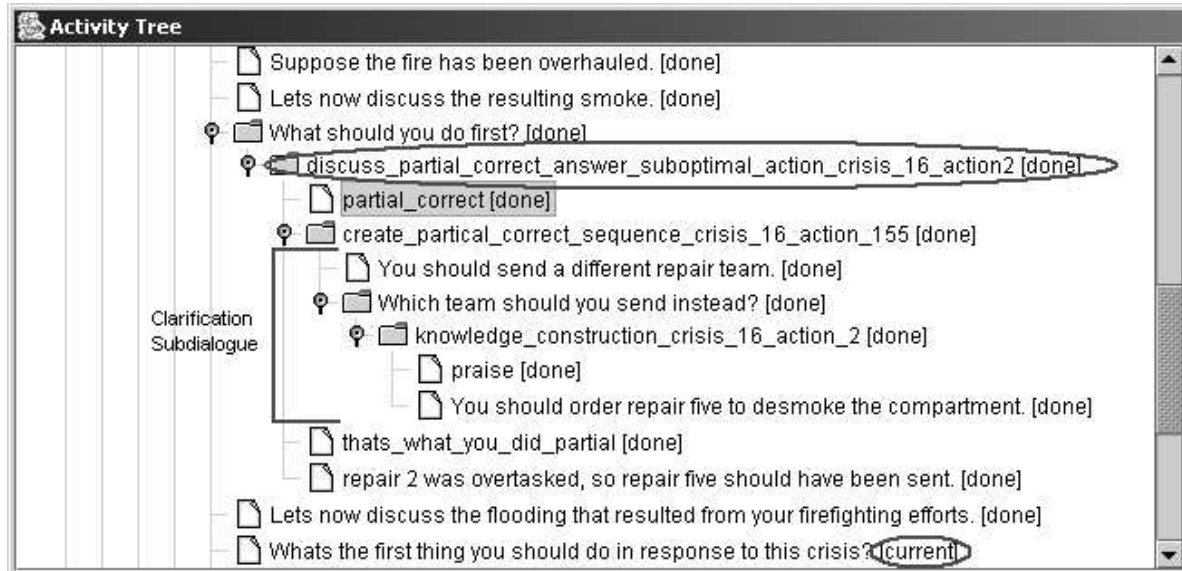
- Acknowledge: Muted Praise
- Hint: Partially Correct Hinting Sequence
- Appraise: Assert Suboptimal Solution Sequence

The following example dialogue illustrates how SCoT-DC uses this tactic in tutorial interactions. In turn (D) the student gives a partially correct answer ("Send repair two to desmoke the compartment")—which is the same as the student's partially correct action from their session with DC-Train, as referenced in (L). In turn (E) the tutor gives an acknowledgement ("Almost"). Turns (F-I) correspond to a hinting sequence for partially correct answers. In turns (J-L), the tutor explains to the student why the solution offered in (D) was not completely right.

- A. Tutor: All right, suppose the fire has been overhauled.
- B. Tutor: Let's now discuss the resulting smoke. [displays grey color in compartment]
- C. Tutor: What should you do first?
- D. Student: Send repair two to desmoke the compartment.
- E. Tutor: Almost.
- F. Tutor: You should send a different repair team.
- G. Tutor: Which team should you send instead?
- H. Student: Repair five.
- I. Tutor: Yes, exactly.
- J. Tutor: You should order repair five to desmoke the compartment.

- K. Tutor: But you sent the wrong repair team during the session.
 L. Tutor: Repair 2 was overtasked, so repair 5 should have been sent.

The tutor's actions are communicated to the Dialogue Manager (see Section 1.1) via the Activity Tree (see Section 1.2) in a domain independent way, as described earlier. This better facilitates not just being able to swap domains with greater ease, but also allows the tutor and Dialogue Manager to be as generic as possible. The figure below demonstrates what the Activity Tree looks like after the example dialogue above.



What is important to get from this picture is that the tutor has laid out for the Dialogue Manager (DM) the structure of the tutorial interaction. The recipe for the tutor's tactic described earlier is apparent in the level just below "discuss partial correct answer suboptimal action". The "partial correct" node corresponds to "Muted Praise" in the tutorial tactic. The node directly below that indicates the partially correct hinting sequence, where the tutor gives a hint ("You should send..."), trying to elicit the correct answer. After the tutor praises the student for giving the correct answer following the hint, the tutor continues with the tactic: first, by asserting that the student's original partially correct answer matched their action during the problem-solving session, and then stating why that was suboptimal. Note that this utterance ("that's what you did") is a couple levels up from the end of the sub-dialogue collaboratively improving the student's partially correct answer. This is because the clarification sub-dialogue is somewhat separate from the subsequent two utterances. The student could have given multiple incorrect answers, causing the clarification sub-dialogue to be quite long. Regardless of how it ends, assuming there is no sudden shift to a different topic, the tutor still wants to establish that the student performed the suboptimal action, and explain why that was suboptimal, but *only* after collaboratively fixing the student's partially correct answer.

Each node which has children is an indication to the DM of a sub-dialogue. For instance, after the last utterance ("Repair two was overtasked...") in the example dialogue, the DM has to move up several levels to "Let's now discuss the flooding...". This is an indication of a topic change, for which the DM will insert a discourse marker "Okay", or "Alright", the same as in line A of the example dialogue. The tutorial interaction following the sub-dialogue initiated by "What should you do first?" will have a similar structure to the sub-dialogue from the node "What's the first thing you should do...". The above picture has no children for that node because the student has not answered the question yet, thus the

tutor will not have chosen a tactic for responding. Once the student gives an answer, based on the classification of that answer and the student's corresponding action in the problem-solving session, the tutor will expand that node with the appropriate tactic (possibly the same tactic as the example dialogue). Finally, although not indicated in the picture, the utterance "Repair 2 was overtasked..." has an indication in it for the DM to pause briefly after speaking the utterance. This pause indicates a tutor-specific desire to allow for the student to ask a question before moving on to the next topic.

3. Natural Language

3.1 Natural Language Understanding

Our approach to natural language input to our system is to interpret each utterance fully as a logical form (LF), a symbolic representation of the utterance's meaning, then to have a subcomponent within the BA extract a domain-specific representation of the relevant information for the system. The linguistic interpretation takes place within the Gemini unification-based natural language understanding system [11] licensed from SRI International.

Gemini uses a grammar written for this particular application. In designing the grammar, we aimed to follow linguistic principles that would allow the grammar to scale up as the application grows and to permit reuse of parts of the grammar in other applications. Thus, in expansions within this domain or in developing a new domain, we would expect to be able to reuse similar LF designs for packaging information. Also, we would expect to be able to reuse syntactic rules for basic constructions, such as transitive and intransitive verb phrases, or question formation, along with the corresponding semantic rules indicating how the meaning of each phrase is constructed compositionally. Another area of reuse would be sets of lexical items, such as the various ways of saying 'yes' or 'no', or the verbs of asking for and reporting information. The domain-specific parts of the grammar include many lexical items. Other domain-specific components are the semantic categories of nouns (e.g., "fire_containment", "valve", "loop_object") which enable verbs to encode selectional restrictions. Also, some syntactic and semantic rules are domain-specific (e.g., those indicating the construction of compartment names or of compartment boundaries).

Our Gemini grammar currently has 147 grammar rules, 669 simple lexical entries, and 929 multi-word lexical entries. Many of the lexical entries are compartment names (there are over 400 compartments on the ship), as well as numbers representing valid frames (port-to-starboard walls), decks, and positions to refer to compartments, compartment boundaries, and valves. There are currently 47 different verbs for distinct actions in the grammar.

3.2 Speech Recognition

One key feature of Gemini for dialogue systems is that Gemini grammars can be automatically compiled to Nuance language models [12]. This feature tightly couples development of the speech recognition and natural language understanding, producing a language model in which every recognized string will have an interpretation. In support of the different states within a dialogue system, Gemini provides the ability to divide a grammar into distinct subset grammars. This division has the greatest use in providing more specialized grammars for context-dependent speech recognition, but it can also be used for

more specialized natural language understanding, to aid in cases where this specialization can eliminate ambiguity.

The process of compiling Nuance language models from Gemini grammars has in practice run into issues of scale, if grammars become highly structurally complex, especially if they are ambiguous. However, large vocabulary sizes in and of themselves have not presented substantial barriers to scaling up. Problems in the compiling process can arise at the point of compiling the Gemini grammar into the Nuance language model, doing a Nuance compile, or in recognition performance. To date, this issue of scale has always been addressable by attention to grammar engineering, without necessitating a compromise in system design, although it signals a possible greater problem area with greater scale.

The potential problems with scale may be addressed by incorporating other kinds of language models. An alternate approach is to design a small, precise grammar-based system, which gains robustness within the larger space of possible user utterances by attempting to guide users toward in-grammar utterances, by recognizing and understanding enough of the out-of-grammar utterances to suggest suitable in-grammar replacements [13].

3.3 Natural Language Generation and Speech Synthesis

Currently, we use templates for generation. Templates do not offer particular advantages for scaling up or reusing systems, but they allow quick development and natural-sounding wording. As a more systematic and principled—thus more reusable—approach, we may consider using the capability of the Gemini natural language system for generation from the same grammar as the understanding grammar in the future.

For speech synthesis, we use Festival [14] with a customized limited domain voice, built with the Festvox [15] tools. The limited domain voice gives higher quality speech for the application, due to recording of prompts for the system with the appropriate intonation, instead of relying on the synthesizer to produce appropriate intonation in a general fashion. However, recording the desired system prompts and preparing the analyzed recordings as input for the Festvox scripts takes a fair amount of time, and as the domain grows in scale, the synthesizer becomes less likely to choose appropriate intonation for novel utterances by selecting appropriate units from the recorded prompts. For a new domain, very little can be carried over from the previous domain, unless there is an overlap of specific system phrases within the set of prompts to be recorded.

A possible midpoint between a custom voice for each domain and using a general synthesized voice is to provide mark-up to a general synthesizer voice for domain-specific elements, particularly given the knowledge inherent in the dialogue manager for template generation or in the grammar for grammar-guided generation. This approach improves synthesis quality a modest, but targeted amount. It should scale reasonably well, with no likely conflicts for additional mark-up, and it should provide opportunities for porting mark-up for similar constructions across domains [16].

4. Knowledge

To understand how domain-specific information may be needed to solve a problem, we have broken it into two major categories: procedural and motivational. Procedural knowledge is the specific knowledge that, given the context of a problem, there is some action the student should take which will move the problem closer to being solved. For example, in the damage control domain, if a Class C fire is known to exist, you should

order a repair team to apply water to it. The predicates (fire) and resulting action (applying water) are obvious, but we are missing the motivational knowledge which tells you why applying water is appropriate (i.e., that applying water to a Class C fire will saturate the fuel source and make combustion considerably more difficult). The main distinction between the two is that procedural knowledge is the functional knowledge actually needed to solve the problem, and you can derive it from the motivational knowledge. Procedural knowledge defines an action-reaction sequence, whereas motivational knowledge gives you the information needed to deduce an action-reaction sequence. Although this distinction permits some overlap, it is generally useful, especially when trying to obtain domain knowledge from experts.

Using both kinds of knowledge, the tutor can attempt to form the best picture of a student's ability within a domain. An automated problem-solver would require only procedural knowledge. Similarly, outside observation of a student solving a problem reflects only procedural knowledge. However, through questioning in interactive dialogue, a tutor can elicit motivational knowledge as well. Therefore, in order to more completely tutor a domain, both kinds of knowledge should exist in the knowledge representation and the tutor should have a mechanism for obtaining both from the student (i.e., observing the student solve a problem and being able to ask questions).

Currently, SCoT-DC only has procedural knowledge, and as with most intelligent tutoring systems, it is represented as a sequence of if-then production rules which define how to react given a new input (e.g., a new fire, running out of water for fire-fighting) [17]. This is the necessary knowledge for solving a problem. The rules are represented as a sequence of clauses, the predicates, and if those are satisfied then a second sequence of clauses, which define modifications to the problem or problem-state, are activated. Observe the example below, which specifies when you should order desmoking given a new input of "fire-extinguished"—in plain English: "after receiving a message for a compartment that the fire in it has been extinguished, if we have a damage report of smoke in that compartment and an unaddressed goal to clear the smoke in that compartment, then using the best repair locker for the compartment, you should desmoke the space."

```
RULE 6324.fire-extinguished.suggest "if there is residual smoke, order desmoking"  
IF    report(find, _, 6801, "Damage Report Fire/Flood/Smoke", [casualty = "smoke", compartment = Compartment])  
      goal(find, unaddressed, 7125, "Clear Smoke", [compartment = Compartment])  
      world-state(find, _, 4302, "Best Repair Locker for Compartment", [compartment = Compartment, station = RL])  
THEN action(create, pending, 5140, "Desmoke space", [compartment = Compartment, target = RL])
```

Note that this is the same knowledge that provides the correct action the tutor would use in the sample dialogue from Section 2.3. The point of conflict in the example dialogue comes from finding the best repair locker for the compartment, which can vary depending on resource availability.

The problem state is represented in terms of three components: problems to be solved, goals which pertain to the solution of the problem, and the specific state of the problem. Given a state, or update to the state, there will be some rule defining the relevant actions to take (such as the rule above). Given the new state, or its update as a result of those actions, there will be further rules defining what to do. Proceeding in this fashion, the problem will eventually be deemed unsolvable (when there are no more possible actions to take), or it will have reached a solution-state. Careful knowledge engineering is very important here to avoid certain pitfalls like infinite loops or ambiguous situations. We do not claim to have a representation for modeling problem solving in *every* domain. This kind of representation will serve well for domains where a student action has a high probability of the 'correct' result on the problem state (e.g., algebra, damage control), but not for domains where the

right solution will involve a lot of intuition or forward/backward reasoning to identify the appropriate next action (e.g., playing Go or Chess).

There are a number of advantages to representing the procedural knowledge in this way. One is that it never needs to be recompiled because it is a static, declarative representation. Many domains which can be defined in terms of problems, related goals, and concrete state variables (actions and events) can be represented in it, at least to the point of knowing how to solve problems (the procedural knowledge). Also, given that it is a functional knowledge representation, and should be able to offer solutions, one can test its strength (or 'ability-level') simply by posing problems and seeing if the knowledge representation can solve them. By doing this, errors in the knowledge should become apparent. They will be easy to debug since for every action taken towards solving the problem only one rule will be fired. We also may have a means for generating justifications of knowledge and actions directly from the knowledge representation, written in syntax that is only a small extension of the knowledge representation syntax itself; this is still being developed [17,18].

By replacing the knowledge from one domain with another, the tutor will not need to be changed. This is where a lot of the effort in porting the system to a new domain would go. Also, a good deal of work will still be required to use new domain-specific language. For the tutor, however, use of the knowledge is the same regardless of the domain. The tactics and strategies communicate with the knowledge abstractly, at the level of problems, goals, states, and student actions. By thinking about a domain at this level, the tutor does not need to know what domain it is tutoring, but only needs to know how to appropriately deal with these objects, as described in Section 2.

Conclusion

In sum, we have presented the components of a scalable, reusable, tutorial dialogue system. A general model of conversational intelligence for dialogue management; a tutoring model whose strategies and tactics are not restricted to any domain; techniques for natural language understanding; and a framework for knowledge representation. All of these attempt to minimize the efforts needed to reuse the tutor for a new domain. The system described in this paper has yet to be fully tested with a variety of users. Currently we are about to begin a user evaluation study to test the robustness of SCoT-DC in classroom and laboratory settings, when facing a variety of different users. In the future we hope to advance the language generation capabilities, extend the model of conversational intelligence, create a robust way of representing motivational knowledge, and use the SCoT architecture as an experimental platform for evaluating different tutoring styles.

References

- [1] Clark, H. H. (1996). *Using Language*. Cambridge: University Press.
- [2] Lemon, Oliver, Alexander Gruenstein and Stanley Peters. (2002). Collaborative Activities and Multi-tasking in Dialogue Systems. *Traitement Automatique des Langues (TAL)*, 43(2):131-154.
- [3] Lemon, Oliver, Anne Bracy, Alexander Gruenstein, and Stanley Peters. (2001). The WITAS Multi-Modal Dialogue System I, In *Proceedings of EuroSpeech 2001*.
- [4] Thomsen-Gray, Zack, Karl Schultz, Brady Clark, Elizabeth Owen Bratt, and Stanley Peters. (2003). Intelligent Tutoring for Non-Deterministic and Dynamic Domains. In *Proceedings of AI-ED 2003*.
- [5] Bulitko, V., and David C. Wilkins. (1999). Automated instructor assistant for ship damage control. In *Proceedings of AAAI-99*.
- [6] Katz, S., G. O'Donnell, and H. Kay. (2000). An Approach to Analyzing the Role and Structure of Reflective Dialogue. *International Journal of Artificial Intelligence in Education*, 11, 320-343.

- [7] Katz, S. and D. Allbritton. (2002). Going Beyond the Problem Given: How Human Tutors Use Post Practice Discussions to Support Transfer. In S. A. Cerri, G. Gouardères, and F. Paraguaçu (Eds.), *Proceedings of 6th International Conference, ITS 2002* (pp. 641-650). Berlin: Springer.
- [8] Freedman, R. (2000). Plan-Based Dialogue Management in a Physics Tutor. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP 2000)*.
- [9] Wilkins, D. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.
- [10] Graesser, A. and N. K. Person. (1994). Question asking during tutoring. *American Educational Research Journal*, 31, 104-137.
- [11] Dowding, J., M. Gawron, D. Appelt, L. Cherny, R. Moore, and D. Moran. (1993). Gemini: A Natural Language System for Spoken Language Understanding. In *Proceedings of ACL 1993*.
- [12] Moore, Robert C. (1998). Using Natural Language Knowledge Sources in Speech Recognition. In *Proceedings of NATO ASI 1998*.
- [13] Hockey, Beth Ann, John Dowding, Gregory Aist, and Jim Hieronymus. (2002). Targeted Help and Dialogue about Plans. In *Proceedings of ACL 2002*.
- [14] Black, A., and P. Taylor. (1997). Festival Speech Synthesis Systems: system documentation (1.1.1). Technical Report HCRC/RT-83, University of Edinburgh Human Communication Research Centre, 1997.
- [15] Black, A., and K. Lenzo. (1999). Building Voices in the Festival Speech Synthesis System (DRAFT) documentation and scripts, <http://www.cstr.ed.ac.uk/projects/festival/papers.html>.
- [16] Bratt, Elizabeth Owen and John Dowding. (2003). Syntactic and Semantic Input to Prosodic Markup in CommandTalk. In *Natural Language Generation in Spoken and Written Dialogue: Papers from the 2003 AAAI Spring Symposium*, Technical Report SS-03-06, AAAI Press, Menlo Park, CA. Pages 1-5, available at <http://www-csli.stanford.edu/~ebratt>.
- [17] Fried, David and David C. Wilkins. (2003). A FOL Knowledge Representation that Supports Expert, Critiquing, and Tutoring Models: DCX 3.0, Knowledge Systems Lab Report UIUC-BI-KBS-2003-0001, Beckman Institute, University of Illinois.
- [18] Fried, David, David C. Wilkins, Eugene Grois, Stanley Peters, Karl Schultz, and Brady Clark. (2003). The Gerona Knowledge Ontology and Its Support for Spoken Dialogue Tutoring of Crisis Decision Making Skills. In *Proceedings of IJCAI 2003*.