# Contextualizing Learning in a Reflective Conversational Tutor

Heather Pon-Barry, Brady Clark, Karl Schultz, Elizabeth Owen Bratt and Stanley Peters
Stanford University
Center for the Study of Language and Information
{ponbarry, bzack, schultzk, ebratt, peters}@csli.stanford.edu

**Abstract**
*Contextualizing learning in an intelligent tutoring system is difficult for many reasons. Goals such as presenting material in an understandable manner, minimizing confusion and frustration, and helping the student reason about their actions all need to be balanced. Previous research has shown reflective discussions (with human tutors) occurring after problem-solving to be effective in helping students reason about their own actions [14]. However, leading a reflective discussion makes it difficult to present information in an understandable manner, and without contextualization it is easy to create student confusion and frustration. This raises the question: how can intelligent tutoring systems effectively contextualize learning in a reflective discussion? In this paper we describe the tutorial architecture of SCoT, a Spoken Conversational Tutor that uses flexible, adaptive planning and multi-modal task modeling to support the contextualization of learning in reflective dialogues.*

## 1. Introduction

One-on-one human tutoring has been established as a highly effective mode of instruction. Studies have shown that students interacting with expert human tutors received test scores 2.0 standard deviations above students in an ordinary classroom setting [3]. In an attempt to approach the effectiveness of human tutors, many developers of intelligent tutoring systems have begun incorporating natural language dialogue into their tutorial systems. While much of the work has focused on using natural language dialogue during the problem-solving session [9, 10, 11], very little work has focused on using reflective dialogue after problem-solving. Many challenges arise when leading reflective discussions after problem-solving that do not come up when leading discussions during problem-solving. For example, students may have a hard time remembering details from the problem-solving session, particularly if the session was complicated. This raises the question: how can an intelligent tutoring system effectively contextualize learning in a reflective discussion?

A reflective, conversational tutor must be able to contextualize information from the problem-solving session in order to successfully talk about past events and to lead a discussion that addresses a student's “trouble spots”. Furthermore, the tutor must use the student's behavior during the dialogue to guide the manner in which new information is presented and to revise planned activities. These issues have motivated the design choices made in our development of the SCoT tutoring system. In this paper, we present the architecture of SCoT's tutorial component and explain how it allows SCoT to facilitate contextualized, reflective, tutorial dialogue.

## 2. Effectiveness of Reflective Tutorial Dialogue

Integrating new information with existing knowledge is a fundamental characteristic of learning. Past research has shown that human tutors can ease this integration process by eliciting self-explanations from the student [5]. Some dialogue-based tutoring systems have taken the approach of eliciting natural language explanations during problem solving [2], but recent studies have provided evidence suggesting that dialogues occurring after problem-solving are especially conducive to student explanations. For example, studies comparing dialogues during problem-solving to reflective dialogues after problem-solving have shown that students are more likely to ask questions and to discuss their reasoning processes in the reflective dialogues [13], and that the reflective dialogues more frequently involved multi-step interchanges between the tutor and the student [16]. In addition, a recent study on the instructional role of reflective dialogue [14] found that students who were asked reflective questions by the tutor learned more (as measured by pre-test and post-test scores) than those receiving no reflection questions.

These results suggest that an intelligent tutoring system supporting reflective dialogue has the potential to be very effective. However, in order to allow the student to integrate new information during a post-session discussion, a reflective tutor must have the capability to contextualize the information it presents. We believe that multi-modal dialogue-based interaction, carried out by a flexible and adaptive planning agent, can aid in this process of contextualization.

## 3. Overview of SCoT

Our approach is based on the assumption that the activity of tutoring is a *joint activity* (in the same way that moving a desk is a joint activity) where the content of the dialogue (language and other communicative signals) is driven by the activity at hand [6]. Following this hypothesis, SCoT's architecture separates conversational intelligence (e.g. turn management, use of discourse markers) from the activity that the dialogue accomplishes (in this case, reflective tutoring). This separation provides for a clearer representation of how and why the nature of a task affects the dialogue.

SCoT-DC, the current instantiation of our tutoring system, is applied to the domain of shipboard damage control. Shipboard damage control refers to the task of containing the effects of fires, floods, and other critical events that can occur aboard Navy vessels. Students carry out a reflective discussion with SCoT-DC after a problem-solving session with DC-Train [4], a fast-paced, real-time, multimedia training environment for damage control. The fact that problem-solving in DC-Train must occur in real-time makes reflective tutorial dialogue more appropriate than tutorial dialogue during the simulation. Because problems co-occur and demand immediate attention, contextualization becomes more difficult.

SCoT is developed within the Architecture for Conversational Intelligence [15], a general purpose architecture which supports multi-modal, mixed-initiative dialogue. SCoT is composed of four separate components: a dialogue manager, a knowledge representation, a student model, and a tutor. These four components are described in sections 3.1 through 3.4.

### 3.1 Dialogue Manager

The dialogue manager handles aspects of conversational intelligence, helping the tutor interpret student utterances in context. It contains multiple dynamically updated components—the two main ones are the *dialogue move tree*, a structured history of dialogue moves, and the *activity tree* (see Figure 3), a hierarchical representation of the past, current, and planned activities initiated by either the tutor or the student. For SCoT, each activity initiated by the tutor corresponds to a tutorial goal; the decompositions of these goals are specified by activity recipes contained in the *recipe library* (see section 4.2).

### 3.2 Knowledge Representation

The knowledge representation provides SCoT a domain-general interface to domain-specific information. In accordance with production-system theories of cognition [1], knowledge specifying causal relationships between events on the ship and proper responses to shipboard crises is encoded in a set of production rules. A knowledge reasoner operates over this production system to provide the tutor with procedural explanations of domain-specific actions, and to provide the student model (see section 3.3) with information about the problem-solving session.

### 3.3 Student Model

The SCoT student model uses a Bayesian network to characterize the causal connections between pieces of target domain knowledge (e.g. a rule for when to perform some action) and observable student actions. Every piece of target domain knowledge has an associated probability representing the system's best guess that the student knows the particular piece of knowledge. This Bayesian framework was chosen because the task of inferring a student's cognitive state from their responses to questions involves a great deal of uncertainty [7]. The student model is dynamically updated during both the problem solving session and the dialogue.

### 3.4 Tutor

The tutor consists of two components: one for planning and executing tutorial activities, and one that contains recipes specifying how to decompose these activities into other tutorial activities or into low-level actions. These components are described in detail in section 4.

## 4. SCoT's Tutorial Architecture

One aspect of leading a reflective discussion is determining how to contextualize information in the most effective manner. Students will likely provide evidence during the dialogue that alters the tutor's original assessment as well as their plan for how to contextualize information. This emphasizes the need for a planning architecture that allows for revisions to the original dialogue plan. The ability to plan and carry out a flexible and coherent dialogue has been a large motivational factor influencing the design of SCoT's tutor component. We have chosen an approach that separates tutorial knowledge (i.e. how to lead a tutorial dialogue) from all other sources of information (e.g. domain knowledge, knowledge of the student). The tutorial knowledge is divided between a three-tier planning and execution system (see section 4.1) and a recipe library (see section 4.2). The three-tier approach to planning and execution was originally developed for artificially intelligent robots and has recently been deployed in tutorial dialogue systems [18]. By separating high-level planning from plan-revision and plan-execution, it allows the tutor to

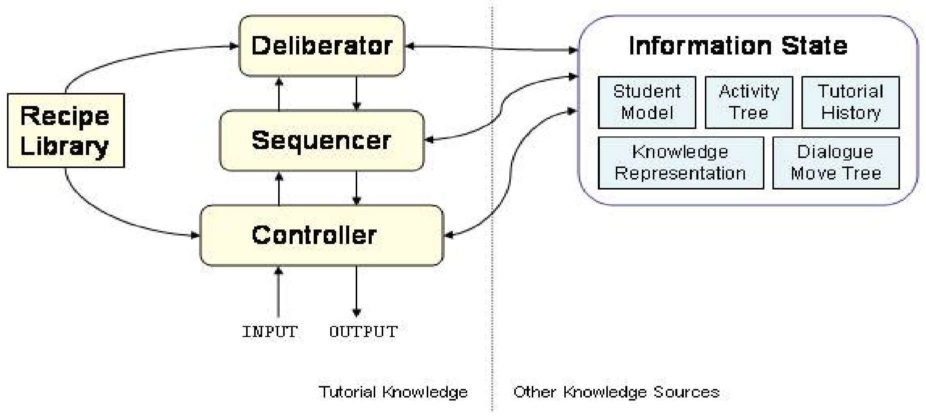

**Figure 1. Interaction between SCoT tutor component & information state.**

lead a flexible dialogue and to continually re-assess information from external knowledge sources in order to appropriately contextualize the information it presents. Figure 1 depicts this separation of knowledge.

## 4.1 Planning and Execution System

The **Information State** encapsulates all external knowledge sources. It provides the tutor with information such as the current context of the dialogue, or a history of the student's interactions with SCoT. By separating tutorial knowledge from other knowledge sources, we not only create a tutor that is reusable in other domains [17], we also provide a clean and consistent way for each component of the tutor to access information pertaining to the current context. Note that Figure 1 only depicts information flow with respect to the tutor—the components of the information state are also updated by one another, and by other components of SCoT.

The **deliberator** makes high-level plans. Before the start of a dialogue, the deliberator uses information gathered from the information state to select a high-level tutorial strategy from the recipe library and place it on the activity tree. During the dialogue, the deliberator can be called on by the sequencer to repair plans (i.e. add or remove planned activities from the activity tree).

The **sequencer** handles plan-revision. Before executing a planned activity, the sequencer checks the current context (from the information state) to make sure that the goals of the planned activity are still relevant, and reorders low-level activities when necessary.

The **controller** handles low-level input to and output from the tutorial component. It initiates the execution of tutorial activities, and handles the classification of student responses (i.e. it interprets logical representations of their utterances as well as actions performed in the GUI).

## 4.2 Recipe Library

An *activity recipe* can be thought of as a tutorial goal and a plan for how the tutor will achieve the goal. The recipe library contains a large body of both high-level and low-level activity recipes. The recipes are written in a scripted language [12] which allows for automatic translation into system activities—the same activities that are stored in the activity tree (each node on the activity tree corresponds to one tutorial goal). Consider the following excerpt from a sample dialogue constructed to highlight the capabilities of SCoT:

|  |  |  |
|---|---|---|
| 1 | Tutor: | OK, in this session there was a fire here, in the supply department storeroom. |
| 2 | Tutor: | You had difficulty controlling this fire, so we should review the various areas of damage control that were covered in this problem. |
| 3 | Tutor: | After the fire alarm sounded, you sent repair 3 to set fire boundaries. However, there is something else you should have done before setting fire boundaries. Can you tell me what that is? |
| 4 | Student: | I'm not sure. |
| 5 | Tutor: | Well, the supply department storeroom is an unmanned space. When the alarm sounds, are you certain that there really is a fire? |
| 6 | Student: | No, I guess not. |
| 7 | Tutor: | Can you tell me now what you forgot to do? |
| 8 | Student: | I should have sent investigators to verify the alarm. |
| 9 | Tutor: | Yes, that's exactly right. |

**Figure 2. Sample dialogue with SCoT**

This dialogue illustrates one tutorial approach to contextualizing the information being presented. The recipe library contains many more, and can easily be augmented. The dialogue excerpt above corresponds to the tutorial goal *discuss_problem_solving_sequence*. After the tutor puts this activity on the activity tree, the system executes the recipe which causes the activity to be expanded into four more-specific activities (i.e. sub-goals). The activity tree in Figure 3 shows this decomposition. Note that, in Figure 3 the activity *situate_problem_context* has also been expanded, but the other three sub-goals are not yet expanded—this diagram represents the state of the activity tree after turn (1) and before turn (2) in the dialogue from Figure 2.

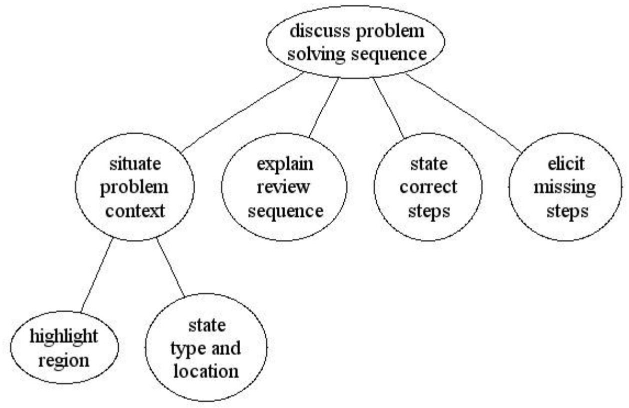

**Figure 3. Sample Activity Tree.**

The tutorial goals (activities) in Figure 3 give rise to a contextualized dialogue in the following ways:

- In turn (1) of the dialogue, it is the tutor's first mention of this problem, so the *situate_problem_context* activity is added to the activity tree, and the tutor describes the type of problem while highlighting its location in the ship display (regions are colored according to the type of crisis, e.g. red for fire, grey for smoke).
- In turn (2) of the dialogue, the tutor tells the student why it chose to review this sequence so that the student will understand the tutor's subsequent turns. This corresponds to the activity *explain_review_sequence*.
- In turn (3) of the dialogue, the tutor contextualizes the problem by reminding the student what they did (they sent repair 3 to set fire boundaries). This corresponds to the activity *state_correct_steps*.
- Also in turn (3), the tutor asks the student what step of the sequence they omitted. Since the student does not provide the information the tutor is looking for (in turn (4)), the tutor provides further information about the context (turn (5)), and reasks the question (turn (7)). This interaction is specified in the decomposition of the *elicit_missing_steps* activity (not shown in Figure 3).

Figure 4 shows simplified recipes for two tutorial activities: *discuss_problem_solving_sequence* and *situate_problem_context*. Each recipe contains three sections: *parameters*, *information to monitor*, and *body*. Parameters specify what information is passed in to the recipe, InformationToMonitor specifies which parts of the information state are used in determining how to execute the recipe, and Body specifies how to decompose the activity into other activities and low-level actions. For example, in the first recipe shown in Figure 4, the tutorial goal *discuss_problem_solving_sequence* is decomposed into either 3 or 4 subgoals (depending on whether the problem has already been discussed). When this recipe is executed, the 4 subgoals are added to the activity tree, and the tutor begins to process their respective recipes.

```
Activity <discuss_problem_solving_sequence> {

  Parameters {
    currentProblem;
  }

  InformationToMonitor {
    currentProblem.alreadyDiscussed;
  }

  Body {
    if (!currentProblem.alreadyDiscussed) {
      situate_problem_context;
    }
    explain_review_sequence;
    state_correct_steps;
    elicit_missing_steps;
  }
}

Activity <situate_problem_context> {
  Parameters {}

  InformationMonitored {
    currentProblem.type;
    currentProblem.location;
  }

  Body {
    highlight_region(currentProblem.location);
    state_type_and_location(currentProblem.type,
      currentProblem.location);
  }
}
```

**Figure 4. Two sample activity recipes.**

Other approaches to contextualization include discussing an analogous hypothetical situation, or exhaustively recreating the details of a problem-solving session. The activity recipe scripting language provides a framework for expressing these tutorial tactics and strategies for contextualization. Furthermore, the modular nature of the recipes makes it easy to test to the effect of different pedagogical and conversational approaches to contextualization.

## 4.3 Multi-modal Interaction

By coordinating spoken and visual output, the tutor has greater flexibility in how it chooses to contextualize information. Both the tutor and the student can interactively perform actions in an area of the graphical user interface called the *common workspace*. In the current version of SCoT-DC, the common workspace consists of a 3D representation of the ship which allows either party to zoom in or out, and to select ("point to") compartments, regions, and bulkheads (lateral walls of a ship). In addition, the common workspace is currently being expanded to incorporate symbolic representations of the changing states of various crises on the ship.

The tutor contextualizes the problems being discussed by highlighting compartments in certain colors (e.g. red for fire, grey for smoke) to indicate the type and location the crises. An example of this coordination can be seen in the way the activity *situate_problem_context* (see Figure 3) is decomposed into both visual and spoken actions. Because the dialogue in SCoT is spoken rather than typed, it has the unique ability to also coordinate the *student's* speech and gesture. This is an area we are currently working on, and once implemented, we hope to support interchanges such as:

| | |
|---|---|
| Tutor: | If there is a fire here [highlights compartment], in compartment 1-126-0-A, which bulkheads should you set fire boundaries on? |
| Student: | I should set primary boundaries here [selects bulkhead], and here [selects other bulkhead]. |

Studies investigating how people combine speech with gestures and diagrams have suggested that participants construct shared models of what they are discussing in order to facilitate communication of difficult content [8]. Allowing the student to explain their reasoning while pointing to objects in the workspace creates a common mode of communication between the student and the tutor, and makes it easier for the tutor to know if the student is contextualizing the problem appropriately. This leads us to believe that multi-modal interaction is extremely helpful in contextualizing reflective tutorial discussions.

## 5. Conclusion

In this paper, we presented the architecture of SCoT's tutor component and described how it uses multi-modal interaction to support the contextualization of learning in a reflective tutorial discussion. By separating tutorial knowledge from domain knowledge and writing activity recipes in a modular way, we have a framework that makes it easy to revise plans as the information state changes and appropriately contextualize the conversation through dialogue and through gesture. We are continuing development efforts to expand the recipe library to address several aspects of conducting reflective tutorial dialogues. One focus is to support self-explanation, in which students use free-form language to explain their own reasoning. A second focus is to round out further tactics for contextualization, and experimentally evaluate their comparative effects.

## Acknowledgments

This work is supported by the Department of the Navy under research grant N000140010660. Further information is available at http://www-csli.stanford.edu/semlab/muri.

## References

[1] Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, NJ: Erlbaum.

[2] Aleven, V., Koedinger, K., & Popescu, O. (2003). A tutorial dialog system to support self-explanation: Evaluation and open questions. In *Proceedings of the 11th International Conference on Artificial Intelligence in Education, AI-ED 2003.*

[3] Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective one-on-one tutoring. *Educational Researcher, 13,* 4-16.

[4] Bulitko, V., & Wilkins., D. C. (1999). Automated instructor assistant for ship damage control. In *Proceedings of AAAI-99.*

[5] Chi, M.T.H., de Leeuw, N., Chiu, M., LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive Science, 18,* 439-477.

[6] Clark, H. (1996). *Using Language.* Cambridge: University Press.

[7] Conati, C.,Gertner, A., & VanLehn, K. (2002). Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction, 12,* 371-417.

[8] Engle, R., Reiner, C., & Lin, W. (In prep.). Establishing mutual understanding by locating models in interactional space. Manuscript in preparation for *Discourse Processes.*

[9] Evens, M., Brandle, S., Chang, R., Freedman, R., *et al.* (2001). CIRCSIM-Tutor: An Intelligent Tutoring System Using Natural Language Dialogue. In *Proceedings of the Twelfth Midwest AI and Cognitive Science Conference, MAICS 2001.*

[10] Freedman, R. (1999) Atlas: A Plan Manager for Mixed-Initiative, Multimodal Dialogue. *AAAI-99 Workshop on Mixed-Initiative Intelligence.*

[11] Graesser, A., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R., & the Tutoring Research Group. (2000). AutoTutor: a simulation of a human tutor. *Journal of Cognitive Systems Research, 1,* 35-51.

[12] Gruenstein, A. (2002). Conversational Interfaces: A Domain-Independent Architecture for Task-Oriented Dialogues. Unpublished M.S. Thesis, Stanford University.

[13] Katz, S., O'Donnell, G., & Kay, H. (2000). An approach to analyzing the the role and structure of reflective dialogue. *International Journal of Artificial Intelligence and Education, 11,* 320-333.

[14] Katz, S., Allbritton, D., & Connelly, J. (2003). Going beyond the problem given: How human tutors use post-solution discussions to support transfer. *International Journal of Artificial Intelligence and Education, 13,* 79-116.

[15] Lemon, O., Gruenstein, A., & Peters, S. (2002). Collaborative activities and multitasking in dialogue systems. In C. Gardent (Ed.), *Traitement Automatique des Langues (TAL, special issue on dialogue), 43(2),* 131-154.

[16] Moore, J.D. (1996). Making computer tutors more like humans. *International Journal of Artificial Intelligence in Education, 7(2),* 181-214.

[17] Schultz, K., Bratt, E., Clark, B., Peters, S., Pon-Barry, H., & Treeratpituk, P. (2003). A Scalable, Reusable Spoken Conversational Tutor: SCoT. In *Proceedings of the AIED 2003 Workshop on Tutorial Dialogue Systems: With a View Towards the Classroom.*

[18] Zinn, C., Moore, J., & Core, M. (2002). A 3-tier planning architecture for managing tutorial dialogue. In *Proceedings of the 6th International Conference, ITS 2002,* 574-584.