BRADY CLARK, OLIVER LEMON, ALEXANDER GRUENSTEIN, ELIZABETH OWEN BRATT, JOHN FRY, STANLEY PETERS, HEATHER PON-BARRY, KARL SCHULTZ, ZACK THOMSEN-GRAY AND PUCKTADA TREERATPITUK

A GENERAL PURPOSE ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS

Abstract. The goal of the Conversational Interfaces project at CSLI is to develop a general purpose architecture which supports multi-modal dialogues with complex devices, services, and applications. We are developing generic dialogue management software which supports collaborative activities between a human and devices. Our systems use a common software base consisting of the Open Agent Architecture, Nuance speech recogniser, Gemini (SRI's parser and generator), Festival speech synthesis, and CSLI's "Architecture for Conversational Intelligence" (ACI). This paper focuses on one application of this architecture - an intelligent tutoring system for shipboard damage control. We discuss the benefits of adopting this architecture for intelligent tutoring.

1. INTRODUCTION

Multi-modal, activity-oriented dialogues with devices present a challenge for dialogue system developers. Conversational interaction in these contexts is mixed-initiative and open-ended. Consider dialogue with an intelligent tutoring system. Dialogue can be unpredictable in tutorial interactions. The student may need to ask the tutor a question; e.g., to request information or request rephrasing. In (1), from Shah et al. (2002, p. 32) a student is requesting information in the form of a yes-no question about the topic in focus. The tutorial domain is physiology.

(1) Student: **Did you count my prediction for sv?**
Tutor: Yes, but you haven't predicted tpr.

In (2), a student is requesting that the tutor rephrase their previous utterance (Shah et al., 2002, p. 34):

(2) Tutor: How are the falls in TPR and in CC connected to decrease in MAP?
Student: **I don't think I understand the question.**
Tutor: What are the determinants of MAP?

Further, the tutor must have a way of reacting to various types of user input; e.g., by adjusting the tutorial agenda when the student asks for clarification about past topics of discussion or when the student asks the tutor to alter the initial overall tutoring plan (e.g., "Can we move on to the next topic?").

In this paper we discuss a new general purpose architecture for intelligent dialogue systems which addresses these issues: the Architecture for Conversational Intelligence (ACI) developed at CSLI. The ACI has previously been used in a dialogue system for multi-modal conversations with a robot helicopter (the WITAS system; Lemon et al., 2002a, 2002b). We focus here on a recent deployment of this architecture in the domain of intelligent tutoring. We will first discuss the intelligent tutoring system we are developing for shipboard damage control. Next, we discuss the ACI for dialogue systems and what benefits it has for intelligent tutoring.

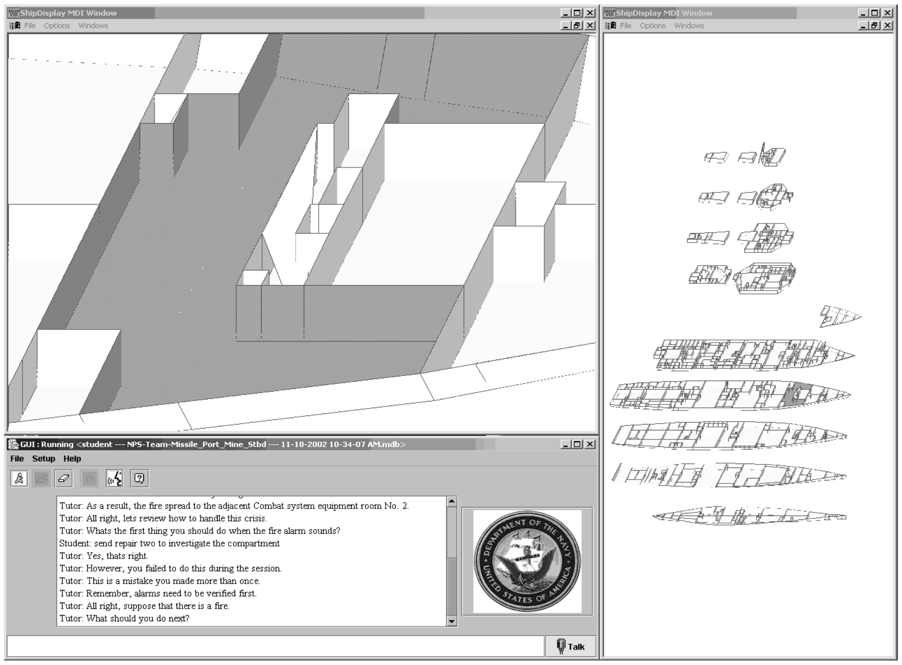
2. AN INTELLIGENT TUTORING SYSTEM FOR DAMAGE CONTROL

Shipboard damage control refers to the task of containing the effects of fire, explosions, and other critical events that can occur aboard Naval vessels. The high-stakes, high-stress nature of this task, together with limited opportunities for real-life training, make damage control an ideal target for AI-enabled educational technologies like intelligent tutoring systems.

We are developing an intelligent tutoring system for automated critiquing of student performance on a damage control simulator (Clark et al., 2001). The simulator is DC-Train (Bulitko and Wilkins, 1999), an immersive, multimedia training environment for damage control. DC-Train's training scenarios simulate a mixture of physical phenomena (e.g., fire) and personnel issues (e.g., casualties).

Recent research has shown that elaborate tutorial interaction during problem solving may be distracting or cause cognitive overload (Katz et al., 2000). This suggests that less may be best in certain learning situations (Sweller et al., 1998, cited in Katz et al., 2000): the cognitive load of simply solving a problem may be high enough that the tutorial interaction should take place after the student has solved the problem (rather than during the problem solving session); i.e., *reflective* tutoring. Further, work by Katz et al. (2000, see also Katz and Allbritton, 2002) has shown that reflective tutoring has a positive effect on learning and enhances the acquisition of strategic and conceptual knowledge. Additionally, we take as our starting point that a tutoring system should model tactics that promote constructive and effortful responses from students (Chi et al., 2001). For these reasons, the intelligent tutoring system we have developed for damage control is *reflective* (the tutor generates plans for post-practice reflection) and *Socratic* (the tutor asks questions rather than giving explanations).

Figure 1 is a screenshot of the graphical user interface for our reflective intelligent tutoring system for damage control. On the right side of the screen is a global view of the ship. Ship compartments are highlighted in coordination with the speech output of the system. The bottom left corner hosts the transcript of the student's conversation with the tutor. The top left corner is the interactive ship display. This part of the GUI is a common workspace for the student and the tutor. The conversational participants can manipulate the ship display; e.g., both the student and tutor can highlight compartments. Thus, the tutor is truly multimodal: it coordinates linguistic input and output (speech) with non-linguistic input and output (the user can indicate a compartment with a mouse click or the system can highlight a compartment).



Tutor: As a result, the fire spread to the adjacent Combat system equipment room No. 2.
Tutor: All right, lets review how to handle this crisis.
Tutor: Whats the first thing you should do when the fire alarm sounds?
Student: send repair two to investigate the compartment
Tutor: Yes, thats right.
Tutor: However, you failed to do this during the session.
Tutor: This is a mistake you made more than once.
Tutor: Remember, alarms need to be verified first.
Tutor: All right, suppose that there is a fire.
Tutor: What should you do next?

Figure 1. Screen shot of GUI for reflective intelligent tutorial dialogue system

(3) provides part of an actual session with our intelligent tutoring system. The tutor runs on a laptop using Windows 2000.

(3) Tutor: Hello, we are about to review your session from yesterday. All right, are you ready?

Student: Yes.

Tutor: In this session there were 5 primary crises and 1 secondary crisis. 5 of the 6 crises were resolved. Most of your actions were correct, however you made several mistakes.

OK, let's discuss a few of those crises. The first crisis was a fire here in Crew Living space No. 1. [*centers on this*

Compartment and highlights it] This fire was not contained early enough. As a result, the fire spread to the adjacent Combat system equipment room No. 2. [*highlights 2nd compartment*]

OK, let's review how to handle this crisis. [*removes highlight from 2nd compartment*] What's the first thing you should do when the fire alarm sounds?

Student: Send repair two to investigate.

Tutor: Yes, that's right. However, you failed to do this during the session. This is a mistake you made more than once. Remember, alarms need to be verified first.

OK, suppose that there is a fire. What's the next step?

Student: Set fire and smoke boundaries.

Tutor: Yes. That's what you did during the session.

OK, what else needs to be done?

Student: The repair team should fight the fire.

Tutor: No. You missed a step. Remember, the crisis might interfere with other ship systems.

Let's try again. What else needs to be done?

Student: I don't know.

Tutor: Well. You should order repair five to mechanically and electrically isolate the compartment. But you didn't do this during the session. This is a mistake you made more than once. Remember, mechanical and electrical isolation protects crucial ship systems.

All right, suppose isolation has begun and boundaries have been set. What's the next step?

Student: Fight the fire.

Tutor: Yes, exactly. That's what you did.

…

Tutor: The next crisis was a flood here in Engine room No. 2. [*highlights compartment*] Let's review how to handle this crisis.

What's the first thing you should do when the flood alarm sounds?

Student: Send repair two to investigate.

Tutor: Almost. Let's review the jurisdiction of repair lockers. The flood alarm was here in Engine room No. 2 [*highlights compartment*]. Please click on the location of the nearest repair locker in the left display above. Let me know when you're ready to continue.

Student: [*clicks on correct compartment*] Please continue.

Tutor: Yes, that's right. Which repair locker is that?

Student: Repair five.

Tutor: Yes, that's right. You should order repair five to investigate the compartment. But you sent the wrong repair team during the session.

The dialogue in (3) matches the 5-step dialogue frame that Graesser and Person (1994) observed in naturalistic tutoring.

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

Figure 2 is the overall architecture of our system (ASR = Automated Speech Recognition, TTS = Text-to-Speech).

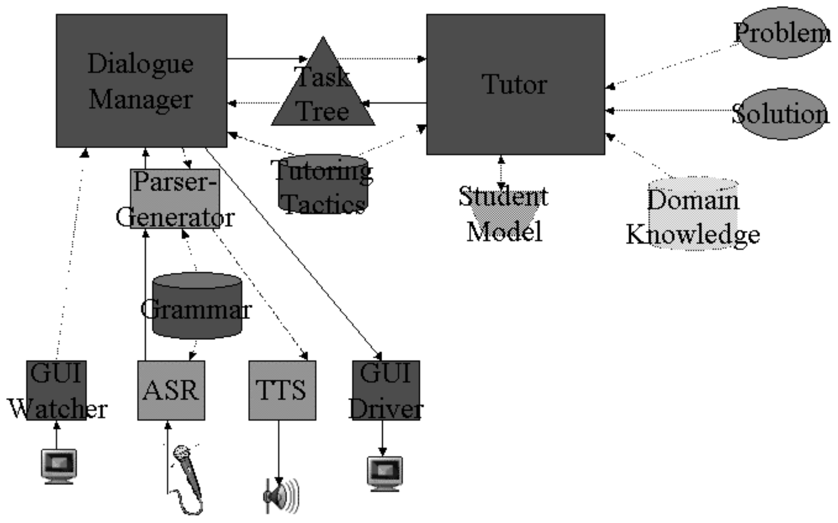


Figure 2. Reflective Tutoring Architecture

In addition to being a Socratic tutor, our tutor shares several features with other intelligent tutoring systems; e.g., CIRCSIM-Tutor (Zhou et al., 1999) and Atlas/Andes (Freedman, 2000).

- *A knowledge base* (problem, solution, and domain knowledge in Figure 2): DC-Train currently encodes all knowledge relevant to supporting reflective intelligent tutoring into a structure called a Causal Story Graph. These expert summaries encode causal relationships between events on the ship as well as the proper and improper responses to shipboard crises.
- *Tutoring tactics* (see Figure 2): To respond to student answers to tutor questions, our tutor draws on a library of tutoring tactics. These tactics are very similar to the plan operators utilized in the Atlas/Andes system. The different components of these tactics are the *object* (what's being taught), the *preconditions* (when a particular tactic can be applied), and the *recipe* (what is the method to be used to teach the object). Preconditions on tutoring tactics involve combinations of a classification of the student's response (e.g., a fully correct or incorrect answer) and actions in their

session with the simulator DC-Train (e.g., an error of omission or correct action).

- *An interpretation-generation component* (ASR, TTS, Grammar, Parser-Generator, and Dialogue Manager in Figure 2): In our system, the student's speech is recognized and parsed into logical forms (LFs). The architecture also allows the tutor's speech to be generated from LF inputs, although we currently use template generation. A dialogue manager inspects the current dialogue information state to determine how best to incorporate each new utterance into the dialogue (Lemon et al., 2002a and 2002b).

An important difference is that CIRCSIM-Tutor and Atlas/Andes are entirely text-based, whereas ours is a *spoken* dialogue system (ASR and TTS in Figure 2). Our speech interface offers greater naturalness than keyboard-based input, and is also better suited to multimodal interactions than keyboard-based input (namely, one can point and click while talking but not while typing). In this respect, our tutor is similar to COVE (Roberts, 2000), a training simulator for conning Navy ships that uses speech to interact with the student. But whereas COVE uses short conversational exchanges to coach the student *during* the simulation, our tutor engages in extended tutorial dialogues after the simulation has ended. An additional significant difference between our system and a number of other intelligent tutoring systems is our use of 'deep' processing techniques. While other systems utilize 'shallow' statistical approaches like Latent Semantic Analysis (e.g. AutoTutor; Wiemer-Hastings et al., 1999), our system utilizes Gemini, a parser/generator based on a symbolic grammar. This approach enables us to provide precise and reliable meaning representations.

As discussed in the introduction, conversation with intelligent tutors places the following requirements on dialogue management (see Lemon et al., 2002b and Clark, 1996):

- *Mixed-initiative*: in general, both the student and the tutor should be able to introduce topics
- *Open-ended*: there are not rigid pre-determined goals for the dialogue

In the next four sections, we discuss in more detail the implementation of our system and how the general purpose architecture for intelligent tutoring systems we have developed meets these two demands.

3. AN ARCHITECTURE FOR MULTI-MODAL DIALOGUE SYSTEMS

To facilitate the implementation of multi-modal, mixed-initiative interactions we use the Open Agent Architecture (OAA) (Martin et al., 1999). OAA is a framework for coordinating multiple asynchronous communicating processes. The core of OAA is a ‘facilitator’ which manages message passing between a number of encapsulated software agents that specialize in certain tasks (e.g., speech recognition).

Our system uses OAA to coordinate the following agents:

- The **Gemini** NLP system (Dowding et al., 1993): Gemini uses a single unification grammar both for *parsing* strings of words into logical forms (LFs) and for *generating* sentences from LF inputs (although, as mentioned above, we do not use this feature currently). This agent enables us to give precise and reliable meaning representations which allow us to identify dialogue moves (e.g., *wh-query*) given a linguistic input; e.g., the question “What happened?" has the (simplified) LF: wh-query(wh([tense(past),action(happen)])).
- The **Nuance** speech recognition server: The Nuance server converts spoken utterances to strings of words. It relies on a language model, which is compiled directly from the Gemini grammar, ensuring that every recognized utterance is assigned a LF.
- The **Festival** text-to-speech system: Festival is a speech synthesizer for system speech output.
- The **Architecture for Conversational Intelligence** (ACI) coordinates inputs from the student, interprets the student's dialogue moves, updates the dialogue context, and delivers speech and graphical outputs to the student (i.e., generation). This agent is discussed in Section 4.

The first three agents are ‘off-the-shelf’ dialogue system components (apart from the Gemini grammar, which must be modified for each application). The ACI agent was written in Java for dialogue management applications in general and is described in more detail in Sections 4 and 5. This OAA/Gemini/Nuance/Festival/ACI architecture has also been deployed successfully in other dialogue systems; e.g., a collaborative human-robot interface (Lemon et al., 2002a, 2002b).

4. ACTIVITY MODELS

An important part of dialogue context to be modelled is the tutor’s planned activities (what topics are going to be discussed and how), current activities (what topic is being discussed) and their execution status (pending, cancelled, etc.). Declarative descriptions of the goal decompositions of activities (COLLAGEN’s “recipes”, Atlas/Andes’ “plan operators”, our “Activity Models”) are a vital layer of representation between the dialogue manager and the tutor.

Intelligent tutoring systems should be able to plan sequences of atomic actions, based on higher-level input; e.g., a problem, the student’s solution to that problem, and domain knowledge. On the basis of this input, the tutor carries out planning (e.g., an initial overall tutoring plan) and then informs the dialogue manager of the sequences of activities it wishes to perform. The model contains traditional planning constraints such as preconditions of actions (as with the tutoring tactics mentioned in Section 2 and discussed in detail below). Dialogue between the tutor and student can be used to revise the *overall tutorial plan* and to update the *student model*. We will briefly discuss both of these possible revisions.

The tutor uses information in an annotated record of the student's performance to construct an *initial* overall tutoring plan; i.e., what problems (e.g., shipboard crises like fires) are going to be discussed. Our tutor currently makes a list of *exemplar* crises that occurred in the student's session with DC-Train. If more than one crisis of a given type occurred, the tutor picks the one with the most errors. The motivation for this particular algorithm is that the student's knowledge and misconceptions will be reflected in the errors they make and that exemplar crises will make for the most interesting dialogues and the most opportunities for learning. This initial overall tutoring plan can be dynamically revised during the tutorial dialogue; e.g., the student can ask to skip discussion of a particular topic by saying (some variant of) "Can we please move on?"

A student model represents the tutor's estimate of what the student knows (or doesn't) know and what skills the student has (or hasn't) mastered. The evidence that is used in constructing a student model is the student's solution to problems, of course, but also the student's interaction with the tutor. For example, the student's answers to the tutor's questions, the student's explanations, and the student's questions to the tutor can all be used to update the student model over time.

We are developing one representation and reasoning scheme to cover the spectrum of cases from devices with no planning capabilities to some with more impressive on-board AI, like intelligent tutoring systems. In the tutor we have developed, both the dialogue manager and the tutor have access to a "Task Tree": a shared representation of planned activities and their execution status. The tree is built top-down by processing a problem, a student's solution to that problem, and the relevant domain knowledge. The nodes of the tree are expanded by the dialogue manager (via the Activity Models specified for the tutor) until only leaves with atomic actions are left for the tutor to execute in sequence. The tutor and the dialogue manager share responsibility for constructing different aspects of the Task Tree; e.g., the dialogue manager marks transitions between topics (with "OK", "All right") while the tutor constructs the overall initial tutoring plan.

Tutoring tactics (e.g., hinting) are one type of Activity Model in our tutor. To initiate a tutoring tactic, the student invokes the tutor by responding to a question. 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 (e.g., Atlas/Andes; Freedman, 2000), each tutorial tactic has a multi-step *recipe* (Wilkins, 1998) composed of a sequence of actions. Actions in a recipe can be primitive actions like providing feedback or complex actions like an embedded tutoring tactic.

An example tutoring tactic is given in Figure 3. For legibility, the key elements are presented in English rather than than Java. The *object* (or *goal*) of the tutoring tactic is to teach the student about an action they failed to perform (an error of omission). This tactic is used when the student answers the tutor's question incorrectly and the student's action in response to a damage event included an error of omission. These are the *preconditions* on the application of this tutoring tactic. For example, in the dialogue in (3) the student said "The repair team should fight the fire" (an incorrect answer and the student forgot to isolate the compartment in their session with DC-Train). The method that the tutor uses to teach the student about their error of omission is given by the *recipe* – a series of sequential system actions.

```
def_strategy discuss_error_of_omission_answer_incorrect
    : goal  (did_discuss_error_of_omission_answer_incorrect)
    : preconditions
        (i) the student's answer is incorrect
        (ii) the student's actions in response to the damage event included an
        error of omission
    : recipe
        (i) provide negative feedback to the student
        (ii) give the student a hint
        (iii) ask a follow-up question
        (iv) classify the student's response
        (v) provide feedback to the student
        (vi) tell the student the rule
        (vii) tell the student that the topic is changing
```

Figure 3. Activity Model for Hinting

The tutor utilizes information in a Causal Story Graph (CSG, an annotated description of the student's performance in their session with DC-Train) to decide which tutorial tactic is appropriate with respect to a student's response to a particular question. For example, in Figure 3, the tutor uses, in the preconditions on the application of the tutoring tactic, the information in the CSG which classifies the relevant action as an error of omission, in addition to the classification of the student's response as an incorrect answer. The preconditions on other tutoring tactics will involve different combinations of action and response classification. Hence, it is the combination of the classification of a student's response (as correct, incorrect, etc.) and action in a DC-Train session (as an error of omission, error of commission, etc.) which determine which tutoring tactic the tutor uses to teach the student.

The tutor then adds a node to the Task Tree describing the tutoring tactic. The tutoring tactic specifies what atomic actions should be invoked (e.g., feedback), and under what conditions they should be invoked. For example, in Figure 3, the tutoring tactic states that that tutor should,

- provide negative feedback to the student
- give the student a hint
- ask a follow-up question
- classify the student's response
- provide feedback to the student

- tell the student the rule
- tell the student that the topic is changing

Nodes on the Task Tree can be active, completed, or cancelled. Any change in the state of a node (e.g., because of a question from the student) is added to the System Agenda (the stack of issues to be raised by the system – see below). For example, in (4), the student asks the tutor a "Why"-question following the tutor's explanation. The tutor responds to the student's question by adding the student's question to the Task Tree and marking it as active. The tutor then asks the student if they would like the tutor to display the relevant section of doctrine. After the student has responded, the node corresponding to the student's question is marked completed.

(4) Tutor: You should request permission from the EOOW to start a firepump. But you didn't do this during the session. Remember, you need the EOOW's permission before starting a firepump.
Student: Why?
Tutor: Well. This is specified in damage control doctrine. Would you like me to display the relevant section of doctrine?
Student: Yes.
Tutor: OK. [*doctrine is displayed*] Let me know when you're ready to continue.

5. DIALOGUE MANAGEMENT ARCHITECTURE

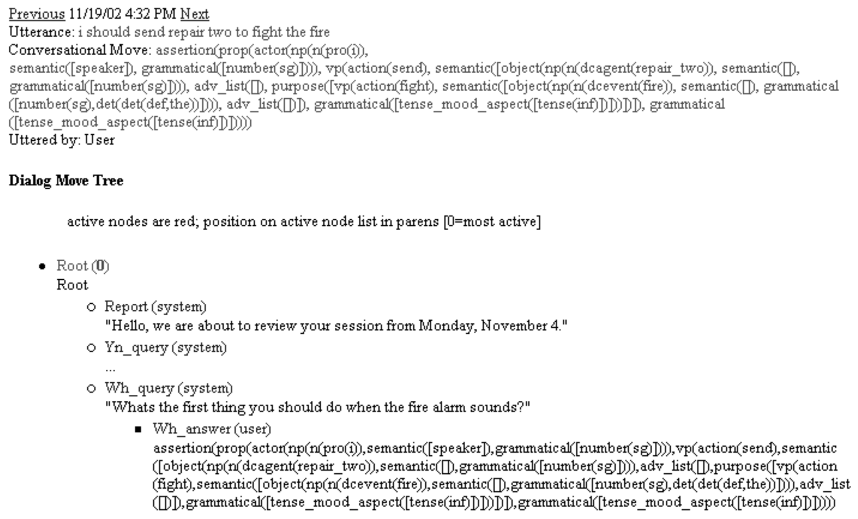
Dialogue Management with the ACI makes use of several recent ideas in dialogue modeling, described in detail in Lemon et al. (2002a, 2002b). Much of what follows in this section is an adaptation of the discussion in Lemon et al. (2002b). The Dialogue Manager creates and updates an *Information State*, corresponding to a notion of dialogue context. Dialogue moves (e.g., *wh-query*, *wh-answer*) update information states. A student's dialogue move might send a response to the tutor, elicit an assertion by the tutor, or prompt a follow-up question. The tutor itself generates dialogue moves that are treated just like the student's conversational contributions.

The ACI includes the following dynamically updated components (see Lemon et al., 2002a, 2002b for full details):

- The **Dialogue Move Tree**: a structured history of dialogue moves and 'threads', plus a list of 'active nodes'
- The **Task Tree**: a temporal and hierarchical structure of activities initiated by the system or the user, plus their execution status
- The **System Agenda**: the issues to be raised by the system

- The **Salience List**: the objects referenced in the dialogue thus far, ordered by recency
- The **Pending List**: the system's questions asked but not yet answered by the student
- The **Modality Buffer**: stores gestures for later resolution

Figure 4 shows an (edited) example of an Information State logged by our system, displaying the interpretation of "I should send repair two to fight the fire".



```
Previous 11/19/02 4:32 PM Next
Utterance: i should send repair two to fight the fire
Conversational Move: assertion(prop(actor(np(n(pro(i)),
semantic([speaker]), grammatical([number(sg)]))), vp(action(send), semantic([object(np(n(dcagent(repair_two)), semantic([]),
grammatical([number(sg)]))), adv_list([]), purpose([vp(action(fight), semantic([object(np(n(dcevent(fire)), semantic([]), grammatical
([number(sg),det(det(def,the))]))), adv_list([])]), grammatical([tense_mood_aspect([tense(inf)])]))])]), grammatical
([tense_mood_aspect([tense(inf)])]))))
Uttered by: User

Dialog Move Tree

        active nodes are red; position on active node list in parens [0=most active]

  • Root (0)
    Root
      o Report (system)
        "Hello, we are about to review your session from Monday, November 4."
      o Yn_query (system)
        ...
      o Wh_query (system)
        "Whats the first thing you should do when the fire alarm sounds?"
          ▪ Wh_answer (user)
            assertion(prop(actor(np(n(pro(i)),semantic([speaker]),grammatical([number(sg)]))),vp(action(send),semantic
            ([object(np(n(dcagent(repair_two)),semantic([]),grammatical([number(sg)]))),adv_list([]),purpose([vp(action
            (fight),semantic([object(np(n(dcevent(fire)),semantic([]),grammatical([number(sg),det(det(def,the))]))),adv_list
            ([])]),grammatical([tense_mood_aspect([tense(inf)])]))])]),grammatical([tense_mood_aspect([tense(inf)])]))))
```

Figure 4. Information State (Dialogue Move Tree)

Dialogue management involves a set of domain-independent *dialogue move types* (e.g., *wh-query*, *wh-answer*, etc.; Ginzburg et al., 2001). A dialogue with the system generates a particular Dialogue Move Tree (DMT). The DMT provides a representation of the current state of the conversation in terms of a structured history of dialogue moves. Each node is an instance of a dialogue move type and is linked to a node on the Task Tree, where appropriate. Further, the DMT determines whether or not user input can be interpreted in the current dialogue context, and how to interpret it.

Incoming logical forms (LFs) are tagged with a dialogue move type. For example, the LF wh-query(wh([tense(past),action(happen)])) corresponds to the utterance "What happened", which has the dialogue move type *wh-query*. How are dialogue moves related to the current context? We use the DMT to answer this question:

- A DMT is a history or "message board" of dialogue contributions, organized by "thread", based on activities. In our tutor, threads correspond to topics like individual ship crises.
- A DMT classifies *which* incoming utterances can be interpreted in the current dialogue context, and which cannot be. It thus delimits a space of possible Information State update functions.
- A DMT has an *Active Node List* (ANL) which controls the order in which this function space is searched.
- A DMT classifies *how* incoming utterances are to be interpreted in the current dialogue context.

A particular Dialogue Move Tree can be understood as a function space of dialogue Information State update functions of the form

$f : \textit{Node} \times \textit{Conversational Move} \rightarrow \textit{Information State Update}$

where *Node* is an *active node* on the dialogue move tree, a *Conversational Move* is a structure (*Input Logical Form*, *Activity Tag*, *Agent*) and an *Information State* is a function $g : IS \rightarrow IS$ which changes the current *IS*. The details of the update function are determined by the node type (e.g., *wh-query*) and the incoming dialogue move type (e.g., *wh-answer*) and its content, as well as the value of the *Activity Tag*.

This technique of modelling dialogue context is a variant of "conversational games" (or "dialogue games"; Carlson, 1983) and, in the context of task-oriented dialogues like tutoring, "discourse segments" (Grosz and Sidner, 1986). Both of these accounts of dialogue context rely on the observation that answers generally follow questions, commands are generally acknowledged, so that dialogues can be partially described in terms of "adjacency pairs" of such dialogue moves. The ACI's notion of *Attachment* embodies this idea.

The two main steps of the algorithm controlling dialogue management are *Attachment* and *Process Node*:

- *Attachment*: processes incoming input Conversational Move c with respect to the current DMT and Active Node List, and "attach" a new node N interpreting c to the tree if possible (i.e., find the most active node on the DMT of which the new node can be a daughter, and add the new node at that location).

- *Process node*: process the new node N, if it exists, with respect to the current information state. Perform an Information State update using the dialogue move type and content of N.

The effect of an update function depends on the input conversational move c (in particular, the dialogue move type and the contents of the logical form) and the node of the DMT that it attaches to. The possible attachments can be thought of as *adjacency pairs* (see Levinson 1983), paired speech acts which organize the dialogue locally. Each dialogue move class contains information about which node types it can attach.

Some examples of different attachments available in the current version of our tutor can be seen in Figure 5 (where activity tags are not specified, attachment does not depend on the sharing of an activity tag, as with the node type *not-recognized*). For example, the fourth entry in the table states that a *wh-query* generated by the tutor, with the activity tag *t*, is able to attach any *wh-answer* by the student with that same activity tag. Similarly, the row for *explanation* states that any explanation by the tutor, with the activity tag *t*, can attach a *why-query* by the student.

Node Type	Activity Tag	Speaker	Attaches
noun-query	t	Tutor	wh-answer(t, user)
why-query	t	Student	report(t, system)
yn-query	t	Tutor	yn-answer(t, user)
wh-query	t	Tutor	wh-answer(t, user)
yn-answer	t	Student	report(t, system)
wh-answer	t	Student	report(t, system)
explanation	t	Tutor	why(t, user)
not-recognized		Student	pardon(system)

Figure 5. Attachment in the Dialogue Move Classes

The possible attachments summarized in Figure 5 constrain the ways in which DMTs can grow, and thus classify the dialogue structures that can be captured in the current version of our tutor. As new dialogue move types are added to the tutor, this table will be extended to cover a greater range of dialogue structures. Note that the tutor's dialogue moves appear on the DMT, just as the student's do.

Recall the requirements placed on automated tutors discussed in Section 2. The DMT structure is able to interpret both the student and tutor input as dialogue moves at any time, thus allowing for *mixed-initiative*. Further, the DMT can handle dialogues with no clear endpoint (*open-ended*). In the next section, we discuss further benefits of the ACI for intelligent tutoring systems, both in the domain of shipboard damage control and in general.

6. BENEFITS OF ACI FOR INTELLIGENT TUTORING SYSTEMS

There are several benefits to CSLI's dialogue management architecture (Lemon et al., 2002b):

- The dialogue management architecture is reusable across domains. As mentioned, the same architecture has been successfully implemented in an unmanned helicopter interface (Lemon et al., 2002a, 2002b). The Activity Models - e.g., the properties of the relevant activities- and (some aspects of) the grammar will have to be changed across domains.

- The Dialogue Move Tree/Task Tree distinction allows one to capture the notion that dialogue works in service of the activity the participants are engaged in. That is, the structure of the dialogue, as reflected in the Dialogue Move Tree, is a by-product of other aspects of the dialogue management architecture; e.g., the Activity Models. The Dialogue Move Tree/Task Tree distinction is supported by recent theories of dialogue; e.g., Clark's (1996) joint activity theory of dialogue.

- The dialogue move types are domain-general, and thus reusable in other domains.

- The architecture supports multi-modality with the Modality Buffer. For example, we are able to coordinate linguistic input and output (e.g., speech) with non-linguistic input and output (e.g., the student can indicate a region of the ship display with a mouse click or the tutor can highlight a compartment).

7. CONCLUSION

We began by identifying two properties of tutorial interaction that a dialogue system must capture: mixed-initiative and open-endedness. We then explained the domain-general modelling techniques we used to build an intelligent tutoring system for damage control. This tutor is novel in that it is the first spoken intelligent tutoring system. Our speech interface offers greater naturalness than text-based intelligent tutors and is better suited to multi-modal interactions. We discussed CSLI's dialogue management architecture and the algorithms we used to develop a tutor that is multi-modal, mixed-initiative, and allows for open-ended dialogues.

7.1. Future Work

We are currently expanding the tutoring module to support a wider range of tutoring tactics and strategies. Some of these tactics and strategies are specific to damage control. Other tactics and strategies are domain-general. We plan to evaluate how well CSLI's dialogue management architecture (the Dialogue Move Tree and the Activity Models we have developed for our tutor) handles tutorial dialogues in other domains; e.g., basic electricity and electronics or algebra.

7.2. Evaluation

As part of the research described here, we have done only informal evaluation of the system so far. Three former damage control assistants, with no previous experience using our tutor, completed a session with our tutor. Each subject was able to complete a dialogue with our system, and data has been collected, including speech recognition error rates. All three dialogues were recorded, and the Information States logged. We are planning two larger evaluation efforts in the near future: one at Stanford University, the other at the Naval Postgraduate School in Monterey, CA. We are also planning several psycholinguistic experiments utilizing our tutor. Broadly, we would like to find out if students learn better with a spoken environment. If so, is that directly because of the naturalness of the modality, or because speech encourages longer utterances and more student initiative?

ACKNOWLEDGEMENTS

This work is supported by the Department of the Navy under research grant N000140010660, a multidisciplinary university research initiative on natural language interaction with intelligent tutoring systems. The Architecture for Conversational Intelligence was funded under the Wallenberg laboratory for research on Information Technology and Autonomous Systems (WITAS) Project, Linköping University, by the Wallenberg Foundation, Sweden.

REFERENCES

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

Carlson, L. (1983). *Dialogue Games: An Approach to Discourse Analysis*. Dordrecht: Reidel.

Chi, M. T.H., S. Siler, H. Jeong, T. Yamauchi, and R. G. Hausmann. (2001). Learning from human tutoring. *Cognitive Science*, *25*, 471-533.

Clark, B., J. Fry, M. Ginzton, S. Peters, H. Pon-Barry, and Z. Thomsen-Gray. (2001). A Multi-Modal Intelligent Tutoring System for Shipboard Damage Control. In *Proceedings of IPNMD-2001*. (pp. 121-125).

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

Dowding, J., J. Gawron, D. Appelt, J. Bear, L. Cherny, R.C. Moore and D. Moran. (1993). Gemini: A natural language system for spoken-language understanding. In *Proceedings of the ARPA Workshop on Human Language Technology*.

Freedman, R. (2000). Plan-Based Dialogue Management in a Physics Tutor. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP 2000).*

Ginzburg, J., I. A. Sag, and M. Purver. (2001). Integrating Conversational Move Types in the Grammar of Conversation. In *BI-DIALOG 2001 - Proceedings of the 5th Workshop on Formal Semantics and Pragmatics of Dialogue*, (pp. 45-56).

Graesser, A. and Person, N.K.. (1994). Question asking during tutoring. *American Educational Research Journal*, *31*, 104-137.

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

Grosz, B. and C. Sidner. (1986). Attentions, intentions, and the structure of discourse. *Computational Linguistics*, *12(3)*, 175-204.

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.

Katz, S. and Allbritton, D. (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.

Lemon, O., A. Gruenstein, A. Battle, and S. Peters. (2002a). Multi-tasking and Collaborative Activities in Dialogue Systems. In *Proceedings of 3rd SIGdial Workshop on Discourse and Dialogue*. (pp. 131-124).

Lemon, O., A. Gruenstein and S. Peters. (2002b). Collaborative Activities and Multi-tasking in Dialogue Systems. In C. Gardent (Ed.), *Traitement Automatique des Langues (TAL, special issue on dialogue)*, *43(2)*, 131-154.

Levinson, S. (1983). *Pragmatics*. Cambridge: Cambridge University Press.

Martin, D., A. Cheyer and D. Moran. (1999). The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, *13*, 1-2.

Roberts, B. (2000). Coaching driving skills in a shiphandling trainer. In *Proceedings of the AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*.

Shah, F., M. Evens, J. Michael, and A. Rovick. (2002). Classifying Student Initiatives and Tutor esponses in Human Keyboard-to-Keyboard Tutoring Sessions. *Discourse Processes*, *32,* 23-52.

Sweller, J., van Merriënboer, J., and Paas, F. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, *10(3)*, 251-296.

Wiemer-Hastings, P. K. Wiemer-Hastings, and A. Graesser. (1999). Improving an intelligent tutor's comprehension of students with latent semantic analysis. In S.P. Lajoie and M. Vivet (Eds.), *Artificial Intelligence in Education (Proceedings of AIED '99)* (pp.535-542). Amsterdam: IOS Press.

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

Zhou, Y., R. Freedman, M. Glass, J. Michael, A. Rovick, and M. Evens. (1999). Delivering hints in a dialogue-based intelligent tutoring system. In *Proceedings of AAAI-99*.