Tools for Authoring a Dialogue Agent that Participates in Learning Studies [1]

Pamela W. JORDAN [a,2], Brian HALL [a], Michael RINGENBERG [a], Yui CUE [b] and Carolyn ROSÉ [b]

[a] *Learning Research and Development Center, University of Pittsburgh*
[b] *LTI, Carnegie Mellon University*

Abstract. TuTalk supports the rapid development of dialogue agents for learning applications. It enables an experimenter to create a dialogue agent with either minimal or no programming and provides the infrastructure needed for testing hypotheses about dialogue. Our main goals in developing this tool were to provide 1) an authoring interface and language for setting up the domain knowledge and resources needed to support the agent and 2) a plug-and-play type of system that facilitates the integration of new modules and experimentation with different core modules. In this paper we describe the authoring tool and the usability studies that have shaped its design, the dialogue that is supported and features of the authoring language and their usage history.

Keywords. linguistics and language technology, authoring tools, intelligent tutoring, dialogue, communication

1. Introduction

TuTalk[3], provides a dialogue system server and authoring tool that supports the rapid development of dialogue systems to be used in learning studies. TuTalk was strongly influenced by our past tutorial dialogue research. As part of this work we created a dialogue system and authoring tools to support our studies involving knowledge construction dialogues (KCDs). A KCD is a main line of reasoning that the tutor tries to elicit from the student by a series of questions. This style of dialogue was inspired by CIRCSIM-Tutor's directed lines of reasoning [1].

TuTalk further extends the types of dialogues supported and adopts an architecture that better supports experimentation. It is intended for two classes of users; 1) non-programmers who intend to test hypotheses about dialogue in learning environments and 2) programmers who wish to measure performance differences of alternative natural language processing (NLP) modules or techniques. While the TuTalk architecture and its NLP modules are not new, a tool that is also tailored to learning applications to ease the

[1]This work was funded by NSF through award number SBE-0354420 to the Pittsburgh Science of Learning Center (PSLC) at Carnegie Mellon University and the University of Pittsburgh.

[2]Email: pjordan@pitt.edu

[3]An acronym for Tutorial Talk.

authoring process is. Thus we combine earlier work from the NLP dialogue community with that of easily authored KCDs for learning applications.

The TuTalk authoring tool enables an experimenter to set up an artificial dialogue agent for their students to interact with during an experiment. The dialogue system server can support multiple dialogue agents and a single dialogue agent can engage in dialogues simultaneously with multiple students. The authoring process consists of specifying how the domain material is to be presented to the students. Depending on how the experimenter sets up the dialogue agent, the students will engage in either agent-led or mixed-initiative dialogues and in either tutorial or conversational style dialogues. Although tutorial and conversational style dialogues are both natural language dialogues, the two are very different in character (see [1] pg. 40 for a brief discussion of the differences).

In this paper we will further describe the authoring tool and the dialogue agents supported by the server. First we provide some background on TuTalk and related dialogue systems. Next, we will give a brief overview of the the dialogues TuTalk supports and then we will discuss the features of the authoring language, their usage history and usability studies that have shaped the design of the authoring tool.

2. Background

Two representative dialogue systems from the NLP community are the DARPA Communicator architecture for spoken dialogue [17] and TrindiKit [11]. Both have modular architectures that support experimentation with dialogue system development. TrindiKit also provides a sophisticated dialogue management module that incorporates a number of dialogue theories [15] and has been used to build a variety of dialogue applications including tutorial dialogue [19]. However in all the applications created with these two architectures, none addressed the problem of non-programmers being able to create a running dialogue application. So far, only two dialogue systems in the AI in Education community have attempted to address this problem of use by non-programmers; our own Atlas system [13,16,7] and the AutoTutor system [3]. A limitation of AutoTutor is that a strategy of hint-prompt-assert is built into the system and the author cannot change this. A problem with Atlas is that it was not modular enough to be a good experimental platform and although dialogue strategy is in the hands of the author, it required programming expertise to adjust some of the general built-in dialogue behaviors; such as determining how to resume after a remediation that may have interrupted the flow of the main line of reasoning.

Atlas was used in the Why2 [16,5], Andes [13] and ITSpoke [12] physics tutoring systems. In addition, it was incorporated into the ProPL computer science tutoring system [10] and it was also used for a separate study of when during physics training dialogues are useful [9]. In total, 13 experiments in 3 science domains have made use of Atlas. All of this work successfully engaged students in natural language dialogues in that students' learn gains as measured by pre and post-tests were significant.[4] TuTalk includes all of the dialogue features of the earlier system, since all were used by the collective set of experiments, and added some new capabilities.[5] However, the dialogue agent

[4]Learning gains and the experimental conditions to which the dialogue agents were compared varied according to the experimental hypotheses being tested. Readers should see the cited papers for details on the results of some of the experiments.

[5]There is an automatic converter available that translates Atlas dialogues into the newer TuTalk format.

is re-implemented with a modular architecture to support experimentation. The architecture comprises a coordinator and a set of replaceable modules (e.g. Natural Language (NL) understanding, NL generation and Dialogue Management) and a dialogue history database. The architecture is described in detail in [8] and will not be addressed further here since our focus for this paper is the dialogue features supported.

3. An Overview of TuTalk Dialogues

The most basic dialogue that one can create with TuTalk can be represented with a finite state machine. Each state contains a single tutor turn. The arcs leaving the state correspond to all possible classifications of student turns. More complex dialogues can be created using two special arcs for calling and returning from a subdialogue. Formally, TuTalk becomes a push-down automaton (PDA) with the addition of these two special arcs because it requires that a stack be added to the finite state machine (FSM). When creating a state, the author enters the text for a tutor's turn and defines classifications for student responses. In the simplest case, a classification is defined by entering text corresponding to how the student might respond.

The NL associated with tutor turns and student responses are derived from concepts. Every concept has a concept label that is used to refer to it within an authored dialogue. A concept in Tutalk is simply a set whose elements are natural language phrases. Ideally each element of the set should represent a similar meaning. It is up to the author to decide what elements have similar meanings. For example the concept label *no* could have a set of elements that all mean a negative response to a yes/no question, such as "no", "nope" and "I guess not.". Concept definitions can be replaced by more complex representations by modifying the concept definition language and replacing the understanding and generation modules with ones that are capable of using the new concept definitions.

The dialogue manager manipulates concept labels only and relies on the NL understanding and generation modules to map between concept definitions and NL strings. Understanding is initiated when the dialogue manager requests a concept classification for an NL input. The dialogue manager supplies the student's NL input and a list of concept labels relevant to the current context to the understanding module. The understanding module returns the concept labels for the concepts that most closely match the student's complete utterance. The default understanding module provided by TuTalk uses a minimum-distance matcher to find the best concept match for an utterance but other means of mapping from NL strings to concepts can be substituted (e.g. LSA, Naive-Bayes, SVM, etc.).

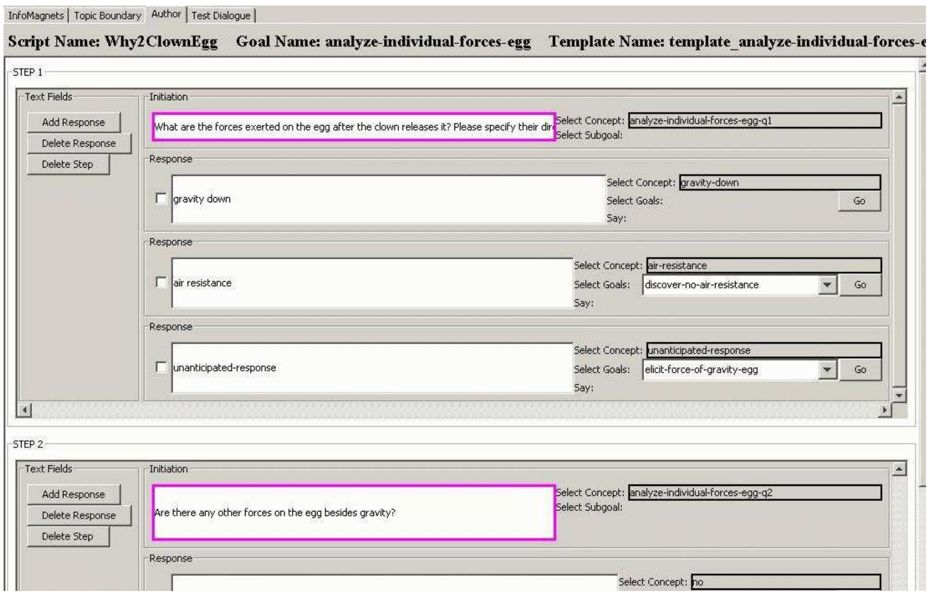
Generation is initiated when the dialogue manager requests that concepts be expressed. The dialogue manager supplies the current discourse context and the labels for the concepts that are to be expressed. Generation merges together phrases appropriate to the concept definitions and the current context and requests that it be output to the student.

TuTalk's dialogue manager is implemented using the reactive planner APE [2] and decides what to express next and how to contextualize student responses. While the dialogue manager does not build-in dialogue strategies, as AutoTutor does, it does build-in some general dialogue behaviors such as refocusing after an interruption and tracking discourse obligations. Discourse obligations are obligations to respond that arise as a result of what a dialogue partner says. The dialogue manager makes use of discourse obligations to resume interrupted topics after a student initiative. Dealing with topic initiative requires more sophisticated stack manipulations than that required for a PDA since an interrupted topic may need to be pruned from the stack and the topic may span multiple stack entries. The dialogue manager also sets up tiers of concept labels (i.e. language models) for concept classification. Concepts that can potentially fulfill an obligation are in the first tier and likely concepts for initiating a new topic are in the next tier.

4. Authoring Dialogues

TuTalk's authoring language is similar to that described in [6,7]. With this language the author specifies a multi-step hierarchically-organized recipe, which is a type of plan structure defined in AI planning, for covering a topic. Recipes address high level goals and are defined as a sequence of any combination of primitive actions and non-primitive recipes [18].

First we will describe the simplest type of recipe and then we will briefly describe the following additional authoring language features; 1) recipe steps that are pointers to subrecipes, 2) responses that indicate follow-up actions, 3) automatic feedback on correctness, 4) management of the dialogue partner's focus of attention, 5) expected responses that require multiple parts to fulfill multiple discourse obligations, 6) annotating knowledge components to assess performance, 7) choosing between alternative recipes, 8) recipe steps that are to be skipped if a specified knowledge component appears in the dialogue history as an effect of some other recipe or step and 9) recipes that are to loop until a knowledge component appears in the dialogue history.



Script Name: Why2ClownEgg Goal Name: analyze-individual-forces-egg Template Name: template_analyze-individual-forces-e

STEP 1

Initiation: What are the forces exerted on the egg after the clown releases it? Please specify their dir
Select Concept: analyze-individual-forces-egg-q1
Select Subgoal:

Response: gravity down
Select Concept: gravity-down
Select Goals:
Say:

Response: air resistance
Select Concept: air-resistance
Select Goals: discover-no-air-resistance
Say:

Response: unanticipated-response
Select Concept: unanticipated-response
Select Goals: elicit-force-of-gravity-egg
Say:

STEP 2

Initiation: Are there any other forces on the egg besides gravity?
Select Concept: analyze-individual-forces-egg-q2
Select Subgoal:

Response:
Select Concept: no

Figure 1. Authoring interface

When writing a recipe to address a topic the author creates one or more steps. The authoring interface in Figure 1 shows one full step with a second step partially visible below it. A primitive step is an initiation that can be optionally paired with a set of anticipated responses. The first step in Figure 1 shows an initiation and a set of expected responses. The specification of an initiation within a step indicates what concept to express or expect in order to achieve initiation of the step. In Figure 1, the concept labels are shown on the far right in the field *select concept* and an NL phrase associated with each is shown in the text boxes in the center. The only difference between an initiation and a response is that only one concept can be associated with an initiation. This is because an initiation corresponds to a state in a FSM and a response to a transition arc.

Subrecipes: Steps that are pointers to subrecipes enable hierarchically organized dialogue. A recipe can embed another recipe by referring to the goal name of that recipe in a step. The effect is that there is a push to the embedded recipe and when the embedded recipe completes, control returns to the parent recipe. All 13 prior experiments made use of this feature.

Follow-up actions: Responses can be authored to include follow-up actions so that vague or flawed responses can be addressed as needed. For example, *discover-no-air-resistance* in Figure 1 is the follow-up recipe that is called when the student response matches the *air-resistance* concept.

A reserved concept label of *unanticipated-response* should always be included as a possible response and an appropriate response recipe specified as with the *elicit-force-of-gravity-egg* follow-up recipe in Figure 1. The *unanticipated-response* arc is followed only if none of the expected response concepts matched. If such a response is not authored and the student's response does not match any of the others specified, then the student's discourse obligation is unfulfilled and the initiation is repeated. The assumption is that the initiation failed and that is why the student didn't respond. Multiple response actions can be indicated and are pursued in the order authored. A reserved action *abort* is now available in TuTalk and causes the parent recipe to end once control is returned to it. All 13 prior experiments used follow-up actions.

Automatic correctness feedback: The system automatically considers whether to provide correctness feedback after every student response. The *say* field on the right side of Figure 1 enables authors to provide their own correctness feedback or transition and overrides the automatic feedback for that response.

Focusing Attention: This is another automatic feature. When resuming an interruption, the dialogue manager heuristically estimates whether the student will need a reminder of the interrupted topic. To remind, it automatically repeats the turn prior to the interruption. In general dialogue this is a less than ideal way to remind the student of the interrupted topic. However, in tutorial dialogue it has the benefit of testing that a remedial interruption was effective since it was found to be a good predictor of learning outcomes [14].

Multi-part Responses: Multi-part responses are analogous to a multiple choice question in which the student needs to select multiple responses to correctly answer. In dialogue, it allows the agent to give the student partial credit and to pursue only what is wrong or missing. When authoring a multi-part response a list of the necessary and sufficient response concepts must be specified for that step. There are follow-up actions for each missing, necessary concept. The reserved concept *unanticipated-response* is selected only if none of the multi-part answer concepts matches. This feature was used in 11 of 13 prior experiments. In two of these experiments it was not needed.

Annotating Knowledge Components for Performance Assessment: Initiations, responses and recipes can all be labelled with optional semantic labels. Only one semantic label per initiation, response or recipe is currently supported. The intent is that items with similar meaning are assigned the same label and can equate to a knowledge component. The dialogue system is able to react to repeated encounters of or alternative actions associated with a knowledge component. Annotated knowledge components are required for the remainder of the dialogue features described to function properly. Because recipes can be hierarchical, a subset of contiguous steps in a parent recipe can bear a knowledge component annotation. This is important because a single step in dialogue frequently does not equate to a knowledge component.

Choosing Between Alternative Recipes: This feature enables an author to create a set of alternatives for a single recipe where the recipe is associated with a targeted knowledge component and is an alternative way of covering that knowledge component. The author optionally assigns each alternative a prerequisite student performance level. The system then assesses the student's previous performance for the knowledge component in order to select an appropriate alternative. If no performance level is supplied, the default behavior is to choose the least recently used recipe when the system is to initiate discussion of the topic. This feature is relatively new and was used in 1 of the 2 experiments in which it was available. In the one experiment, 11% of the recipes had alternatives with assigned performance levels.

Optional Recipe Steps: The optional step feature allows a knowledge component, when it is about to be covered in the dialogue, to be skipped. It checks whether the semantic label on the initiation appears in the recent dialogue history and was successfully covered at that time. It the optional step is a subgoal, the semantic label for the subrecipe is checked. This feature is relatively new and was used in 1 of 2 experiments for which it was available and is now an important feature in an experiment that is currently under development. See [7] for details on optional steps and choosing between alternatives. Figure 2 shows a multi-part answer (1), an optional step (2) and follow-up actions (3).

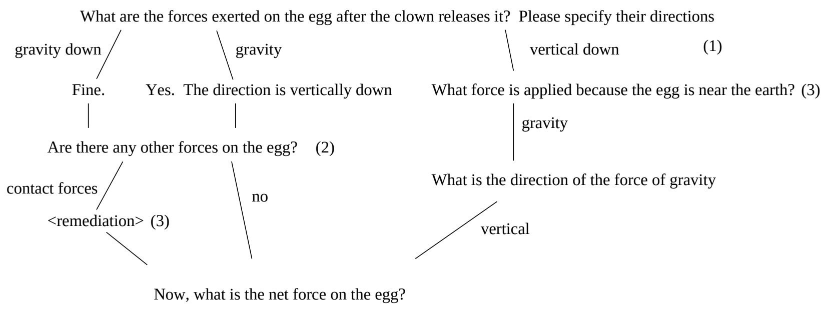

What are the forces exerted on the egg after the clown releases it? Please specify their directions
gravity down
gravity
vertical down
(1)
Fine.
Yes. The direction is vertically down
What force is applied because the egg is near the earth? (3)
gravity
Are there any other forces on the egg? (2)
What is the direction of the force of gravity
contact forces
no
<remediation> (3)
vertical
Now, what is the net force on the egg?

Figure 2. The paths of three Why2-Atlas students for the same recipe

Looping: The scripting language also makes use of semantic labels as termination conditions to implement looping of an entire recipe. All semantic labels included in the termination condition must appear in the dialogue history before the loop will terminate. Only *ands* of labels is supported. If an author wants only some steps in a recipe to loop, then this can be accomplished by extracting those steps into a subrecipe and setting the loop for just that subrecipe. Currently no expiration is set for semantic labels in the case of loops. This is a new feature of TuTalk only. A previous experimenter asked about such a capability as did a prospective TuTalk user. As we see how authors use this feature we will address the issue of setting expirations on loop conditions.

5. Usability Study

The TuTalk authoring environment has been designed and developed using an iterative, user centered design methodology. An earlier paper [4] describes how our initial design for the authoring environment was based on a variety of small pilot studies and user observations in connection with earlier authoring environments. The first fully functioning prototype was then tested in Summer 2005 by 10 users with varying levels of technical expertise and no previous experience authoring dialogues. We provided the users with a short set of training materials and training tasks, which we guided them through. This required about 15 minutes. We then administered a paper-based quiz in which we tested users on the basic functionality of various interface elements in the environment. Finally, we presented the users with a second set of tasks to perform independently in the environment. A month later we tested users again to see how much knowledge they retained over time and then retrained and retested them. On average users acquired about 80% of what we exposed them to based on the immediate assessment. This dropped to 60% after one month, but increased to 85% after retraining. The most striking finding was that users seemed to have difficulty grasping the concept of the stack based structure of the authored dialogues. Observations from these studies informed the development of the environment that we released for use at the 2006 Pittsburgh Sciences of Learning Center summer school. While all participants at the summer school had some hands-on training with the TuTalk authoring environment, 9 participants used the TuTalk tools to develop an intelligent tutoring system prototype.

Past and present users of TuTalk were selected by their projects to be authors based on their domain expertise and not their knowledge of programming. Their experience ranged from none at all (2), to programming web pages (1), to people with extensive knowledge of programming. Beyond observations of use at the summer school, we have conducted in-depth observations of use in the past year and are continuing to refine the design of the authoring environment.

6. Conclusion

We have described the TuTalk dialogue agent and its authoring tool. TuTalk provides a plug-and-play type of dialogue system for learning applications that facilitates integration with new modules and experimentation with different NLP modules and an authoring language for setting up the necessary domain knowledge and resources. The 13 experiments in which the predecessor to TuTalk was used, were all successful and demonstrated that non-programmers can use it. A new experiment has demonstrated that modules within TuTalk can be replaced and supplemented. This experiment replaced the NL understanding module with a human interpretor.

We welcome new users to download and use the authoring tool.[6] It runs as a client

[6]http://andes3.lrdc.pitt.edu/TuTalk/

application and by default uses the dialogue system server that we are hosting. The dialogue system server source is also available so that researchers can set up their own servers and modify the agents it supports to meet their requirements.

References

[1] M. Evens and J. Michael. *One-on-One Tutoring by Humans and Computers*. Lawrence Erlbaum Associates, Inc., 2006.

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

[3] A. C. Graesser, K. Wiemer-Hastings, P. Wiemer-Hastings, R. Kreuz, and the TRG. Autotutor: A simulation of a human tutor. *Journal of Cognitive Systems Research*, 1:35–51, 1999.

[4] G. Gweon, J. Arguello, C. Pai, R. Carey, Z. Zaiss, and C.P. Rosé. Towards a prototyping tool for behavior oriented authoring of conversational interfaces. In *Proceedings of the ACL Workshop on Educational Applications of NLP*, 2005.

[5] P. Jordan, M. Makatchev, U. Pappuswamy, K. VanLehn, and P. Albacete. A natural language tutorial dialogue system for physics. In *Proceedings of the 19th International FLAIRS conference*, 2006.

[6] P. Jordan, C. Rosé, and K. VanLehn. Tools for authoring tutorial dialogue knowledge. In *Proceedings of AI in Education 2001 Conference*, 2001.

[7] P. W. Jordan, P. Albacete, and K. VanLehn. Taking control of redundancy in scripted tutorial dialogue. In *Proceedings of the Int. Conference on Artificial Intelligence in Education, AIED2005*. IOS Press, 2005.

[8] P. W. Jordan, M. Ringenberg, and B. Hall. Rapidly developing dialogue systems that support learning studies. In *Proceedings of ITS06 Workshop on Teaching with Robots, Agents, and NLP*, 2006.

[9] S. Katz, J. Connelly, and C. L. Wilson. When should dialogues in a scaffolded learning environment take place? In *Proceedings of EdMedia 2005*, volume 65, 2005.

[10] H. C. Lane and K. VanLehn. Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education*, 15(3):183–201, September 2005.

[11] S. Larsson and D. Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340, 2000.

[12] D. J. Litman and K. Forbes-Riley. Predicting student emotions in computer-human tutoring dialogues. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain, July 2004.

[13] C. Rosé, P. Jordan, M. Ringenberg, S. Siler, K. VanLehn, and A. Weinstein. Interactive conceptual tutoring in atlas-andes. In *Proceedings of AI in Education 2001 Conference*, 2001.

[14] M. Rotaru and D. Litman. Exploiting discourse structure for spoken dialogue performance analysis. In *Proceedings of the Empiricial Methods in Natural Language Processing Conference*, 2006.

[15] D. Traum and J. Allen. Discourse obligations in dialogue processing. In *ACL94, Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 1–8, 1994.

[16] K. VanLehn, P. Jordan, C. Rosé, D. Bhembe, M. Böttner, A. Gaydos, M. Makatchev, U. Pappuswamy, M. Ringenberg, A. Roque, S. Siler, and R. Srivastava. The architecture of Why2-Atlas: A coach for qualitative physics essay writing. In *Proceedings of Intelligent Tutoring Systems Conference*, volume 2363 of *LNCS*, pages 158–167. Springer, 2002.

[17] M. Walker, A. Rudnicky, R. Prasad, J Aberdeen, E. Bratt, J. Garofolo, H. Hastie, A. Le, B. Pellom, A. Potamianos, R. Passonneau, S. Roukos, G. Sanders, S. Seneff, and D. Stallard. DARPA communicator: Cross-system results ofr the 2001 evaluation. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, 2002.

[18] R. Michael Young, Martha E. Pollack, and Johanna D. Moore. Decomposition and causality in partial order planning. In *Second International Conference on Artificial Intelligence and Planning Systems*, 1994. Also Technical Report 94-1, Intelligent Systems Program, University of Pittsburgh.

[19] C. Zinn, J. Moore, and M. Core. Intelligent information presentation for tutoring systems. In O. Stock, editor, *Multimodal Intelligent Information Presentation*. Springer, 2005.