Rapidly Developing Dialogue Systems that Support Learning Studies

Pamela Jordan, Michael Ringenberg and Brian Hall
Learning Research and Development Center
University of Pittsburgh
{*pjordan,mringenb,mosesh*}*@pitt.edu*

Abstract: We describe a dialogue system construction tool that supports the rapid development of dialogue systems for learning applications. Our goals in developing this tool were to provide 1) a plug-and-play type of system that facilitates the integration of new modules and experimentation with different core modules 2) configuration options that effect the behavior of the modules so that the system can be flexibly fine-tuned for a number of learning studies and 3) an authoring language for setting up the domain knowledge and resources needed by the system modules.

Keywords: dialogue system, learning environments

INTRODUCTION

In this paper we introduce, TuTalk[1], a dialogue system shell and content scripting language that supports the rapid development of dialogue systems to be used in learning studies. We intend to make TuTalk publically available once development is completed.

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. Typically at the end the tutor summarizes the main line of reasoning. This style of dialogue was inspired by CIRCSIM-Tutor's directed lines of reasoning [Evens and Michael, 2006]. The KCD dialogue system was used in the Why2-Atlas [VanLehn et al., 2002, Jordan et al., 2006], Andes [Rosé et al., 2001] and ITSpoke [Litman and Forbes-Riley, 2004] physics tutoring systems. In addition, it was incorporated into the ProPL computer science tutoring system [Lane and VanLehn, 2005] and it was also used for a separate study of when during physics training dialogues are useful [Katz et al., 2005]. All of this work successfully engaged students in natural language dialogues and students were able to learn from the interactions. The goal for TuTalk was to redesign this software to provide 1) a modular architecture 2) easily modifiable system behavior to support a larger variety of learning studies and 3) easily authorable content.

We anticipated addressing three classes of users; 1) those who intend to test complex dialogue hypotheses (e.g. is a speech-act pattern of inform->justify better than justify->inform for learning a concept) 2) those whose central hypothesis may not be about the fine points of dialogue but may nonetheless need a natural language dialogue capability to support their learning experiment (e.g. when will a natural language dialogue-like capability increase learning) 3) those who wish to measure performance differences of alternative natural language processing (NLP) modules or techniques (e.g. what effect does improved merging of sentences during language generation have on student comprehension).

Two problems with experimenting with dialogue systems for the first two classes of users are 1) getting the desired configuration and system behavior necessary to support an experiment that tests a particular hypothesis and 2) setting up the domain specific resources required by the dialogue system to support the experiment. No one set of NLP modules or instantiation of an NLP module will be perfect for providing all the dialogue behaviors that an experimenter may need. But by adding some finer-grained control over the behaviors of modules, one set of dialogue system modules may support many experiments before other modules must be swapped in to provide the desired dialogue system behavior.

While our goals of a plug-and-play architecture and easy configuration are not new to the NLP dialogue community, a tool that is also tailored to learning applications to ease the authoring process

[1] An acronym for Tutorial Talk.

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

In this paper we will first briefly describe the architecture. We will then describe the authoring language, the options it offers and how configuration choices can alter the interpretation of these options and thus the behavior of the system. We will conclude with some tentative plans for learning experiments with this new dialogue system building tool.

SYSTEM ARCHITECTURE

TuTalk is structured as a collection of core dialogue system modules and resources that together form an end-to-end tutorial dialogue system. The modules have well-defined APIs in order to support the addition of new modules with only minimal changes to the system and allow core modules to be replaced or upgraded as different learning applications demand different functionality or improved approaches.

Modules act as agents and can advertise themselves as offering particular services and can issue requests for other services. Service requests are dispatched to appropriate modules by the Coordinator module which manages a task list. It adds requests to the list and routes requests from the list.

In addition to the Coordinator, the modules currently implemented for the system include, among others, the Input, Output, Logger, Understanding, Generation, Student Model, Dialogue Manager and Dialogue History Manager modules. We will focus our descriptions of the modules on just the Understanding, Generation, Dialogue Manager, Student Model and Dialogue History Manager modules.

Understanding and Generation

Understanding and generation make use of author defined concepts. 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 identifiers relevant to the current context to the Understanding module. The Understanding module returns the concept identifiers for the concept definitions 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. However, the modular architecture allows the Understanding module to be easily replaced with any natural language classification approach (e.g. LSA, Naive-Bayes, SVM, etc.). In addition to concept classification, Understanding is also supported by a Normalizer module which addresses stemming and spelling correction for English. The Normalizer module can be replaced by one that supports a language other than English in order to support language learning studies.

Generation is initiated when the Dialogue Manager requests that concepts be expressed. The Dialogue Manager supplies the transitioning context and the identifiers for the concepts that are to be expressed. Generation merges together phrases appropriate to the concept definitions and the current discourse context and requests that it be output to the student. If alternatives are available Generation will request guidance from modules that advertise the ability to provide such advice, such as the Student Model.

Dialogue Manager

TuTalk's dialogue manager is loosely characterized as a finite state network with a stack that is implemented using the reactive planner APE [Freedman, 2000]. Finite state approaches are appropriate for dialogues in which the task to be discussed is well-structured and the dialogue is to be system-led [McTear, 2002], as was the case for all the earlier learning applications that used KCDs. TuTalk retained the finite state dialogue management approach in order to keep the demands on the author as low as possible. However, TuTalk adds simple options to the authoring language that are meant to help maximize adaptability to the dialogue context and meet the needs of learning applications. These simple options, in conjunction with differences in the dialogue manager and its built-in knowledge about conducting a dialogue and interpreting the scripting language, help mitigate some of the limitations of finite state approaches without requiring the author to make difficult knowledge engineering decisions (e.g. the tutorial intention for presenting a given knowledge component in the current discourse context) or to program when changes to the behavior of the system are needed. One of the changes made was to add tracking of discourse obligations [Traum and Allen, 1994] which in part supports student topic initiative during dialogue. With the addition of obligation tracking the system now incorporates all but the common ground aspect of the information state defined by TrindiKit [Traum et al., 1999].

A state in the finite state network is either a push to a sub-network as with the right-most and left-

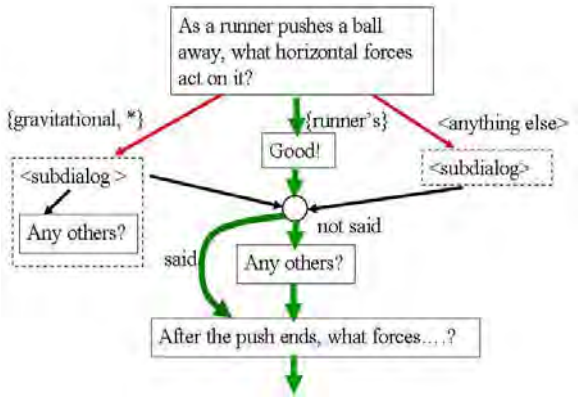

Figure 1: Finite State Network with response concepts and optional steps.

most nodes in Figure 1 or an initiation that expresses some concept plus an optional response as with the top node and its three branches in Figure 1. There is a sub-network for each complex topic to discuss in dialogue. When a state does set up a discourse obligation (e.g. tutor asks a question as with the top node in Figure 1), there is a set of anticipated concepts to recognize for each conceptually different satisfactory and unsatisfactory response. Which concept a response expresses is part of the dialogue context that can help decide the next state to which to move. Thus a response concept can help select an arc between two states in the network. Responses that correspond to unsatisfactory concepts lead to a state that is a push to a subnetwork that addresses the unsatisfactory response. These subnetworks are written to anticipate an eventual return to the next state in the parent network. By default, if a state does not setup an obligation for a response then the transition is to the next state in the network.

In addition to response concepts, three additional conditions can be used in deciding which state to go to next. One is a test to skip over a state if the content of that state is already in the discourse history as with the *said* and *not said* arcs in Figure 1. The second transition condition is a test of which next state is appropriate for a student according to the student's past performance. For example, there could be an alternate state relative to the last node in Figure 1. The third transition condition is just before a pop from a remediation sub-network and tests that the state before the push is still in the student's focus of attention according to the dialogue history. If it is not in the student's focus of attention then the tutor turn before the push is repeated and otherwise the pop is completed.

A higher level of dialogue management above the finite state network tracks discourse obligations and makes use of them to address student topic initiatives. It sets up tiers of concept identifiers (i.e. language models) for concept classification with concepts that can potentially fulfill an obligation being in the first tier and likely concepts for initiating a new topic in the next tier. In addition, it interprets the scripting language and causes the system to implement these scripted dialogues according to the configuration choices made by the dialogue system author. For example, it checks the settings of configuration options in the preconditions of the APE operators that implement each option.

Student Model and Dialogue History Manager

The Student Model and Dialogue History Manager modules provide limited assessments of student performance that the Dialogue Manager uses in making decisions about dialogue. The Dialogue History Manager records information about what concepts have been conveyed by the system and what concepts have been expressed by the student. In addition, the system also records a truth value for concepts the student has expressed when such a value can be determined. This information is used by the Dialogue Manager to decide what type of immediate feedback to give to a student (e.g. positive feedback for concepts that are true in the current context). The Dialogue History is also used to provide rough assessments of student performance with respect to a particular concept when 1) the concept is covered multiple times, 2) there are options for how to cover the concept and 3) there is a truth value for the concept in the current context. For example, in the Why2-Atlas system we used this type of assessment to decide whether to cover concepts in a given context by asking questions of either increasing or decreasing difficulty of students. On the first attempt to cover a concept, the system can assert it and on the second it can hint at it and when attempting to cover it again much later it can ask an open-ended question that should elicit the concept if the student has previously demonstrated good understanding of the concept

each time.

AUTHORING CONCEPTS AND DIALOGUE SCRIPTS

Authoring a dialogue is like writing a movie script with many different endings. TuTalk's scripting language is similar to that described in [Jordan et al., 2001, Jordan et al., 2005] and allows the author to specify a multi-step hierarchically-organized recipe (a type of plan structure defined in AI planning) for covering a topic. Recipes are high level goals that are defined as a sequence of any combination of primitive actions and non-primitive recipes [Young et al., 1994]. In terms of a finite state network, a step in a recipe equates to a state and a state comprises a concept that is to be expressed and the context that is considered when selecting the next state to which to transition. The most influential part of the context for selecting the transition is the anticipated responses to the concept that was just expressed. The author indicates concepts to be expressed and expected responses to it by using concept identifiers when writing a recipe.

In addition to authoring dialogue content, there are a number of built-in policies that guide the behavior of the dialogue system. There are policies for how to choose between alternative recipes, for choosing between alternative ways of expressing a concept, for assessing focus of attention, for skipping states and for handling initiative. For each policy, a number of alternative policies have been defined and the author can choose which alternatives to use as the default at system configuration time. Some of the default policies can be overridden within individual recipes while authoring the recipe. For example, the policy for deciding to skip a state can be overridden by indicating that the semantic identifier associated with the state only has to appear once in the dialogue history while the default policy may take into account focus of attention.

First we will describe how concepts and recipes are authored. Then we will discuss how the system handles initiative.

Defining Concepts

A concept in Tutalk is simply a set whose elements are either natural language phrases or a string of words. Ideally each element of the set should represent a similar meaning. Every concept has a concept identifier that is used to refer to it when specifying recipe steps and it must be a unique identifier. It is up to the author to decide what elements have similar meanings. For example the concept identifier *no* could have a set of elements that all mean a negative response to a yes/no question, such as "no", "nope" and "I need a few minutes". Concept definitions can be replaced by more complex representations by modifying the concept definition language and providing Understanding and Generation modules that are capable of using these more complex concept definitions.

A number of phrase elements for a concept can be combined by including only stems and non-function words (e.g. the concept element "I not appetizer" would cover such phrases as"I do not want an appetizer", "I would not like any appetizer") When an element is not a complete phrase or NL expression then it is marked as being a minimal phrase. Full NL phrases are stripped of morphology as necessary during understanding and how this is done depends upon the natural language in which the element is expressed (e.g. French or English).

```
ask_share_appetizer
     [level=1, So, should we share an appetizer?]
     [level=2, I'd like to share an appetizer.  What looks good to you?]

skip_appetizer
     [phrase=minimal, I not appetizer]
     [lang=fr, Je n'ai pas le temps pour un amuse-gueule]
     [processing-type=extended, principle1 principle2]
```

Figure 2: An example concept specification

When the system is configured for an application, the default language for concept elements is set and any additional languages that are to be used are listed. Concept elements that are not expressed in the default language are marked with which language it is, using its two-character ISO 639.1 language code[2]. For example, in the concept definition for `skip-appetizer` in Figure 2, the phrase *Je n'ai pas le*

[2]http://www.w3.org/WAI/ER/IG/ert/iso639.htm

temps pour un amuse-gueule is flagged as French.

The author can optionally assign a level of difficulty to each phrase element of a concept definition. Then the Student Model can be queried to determine which level is the right choice for expressing a concept to a student. Difficulty levels are only relevant for non-minimal elements since minimal ones cannot be used by the system to express a concept.

Specific concept elements may require different or additional processing beyond what the NLU module provides. For example, in the Why2-Atlas system [Jordan et al., 2006], long explanations required special processing as indicated by `processing-type="extended"` in Figure 2. In this example, the long explanation elements, *principle1 principle2*, are actually pointers to complex knowledge representations. The dialogue in Figure 3, is tagged for three different types of processing that was used for concept recognition in Why2-Atlas.

T: Do you know a principle that relates net force and acceleration? If you do, please define it for me
S: f=ma [equation identifier]
T: Fine. Using this principle, what is the value of the horizontal component of the acceleration of the egg? Please explain your reasoning.
S: Zero because there is no horizontal force acting on the egg. [explanation classifier]
T: Very good. At this point,...
.
.
T: How does the egg's horizontal average velocity during its flight compare to its horizontal initial velocity?
S: Equal [short answer classifier]

Figure 3: A verbatim Why2-Atlas dialogue with special understanding processing marked

Recipes and steps

First we will describe the simplest type of recipe and then we will briefly describe the following additional capabilities that the authoring language provides; 1) recipe steps that are pointers to sub-recipes, 2) responses that indicate follow-up actions, 3) expected responses that require multiple parts to fulfill multiple discourse obligations, 4) recipe steps that are to be skipped if a specified semantic identifier appears in the dialogue history as an effect of some other recipe or step, 5) specifying alternatives and 6) recipes that are to loop until a semantic identifier appears in the dialogue history as an effect of executing some other recipe or step.

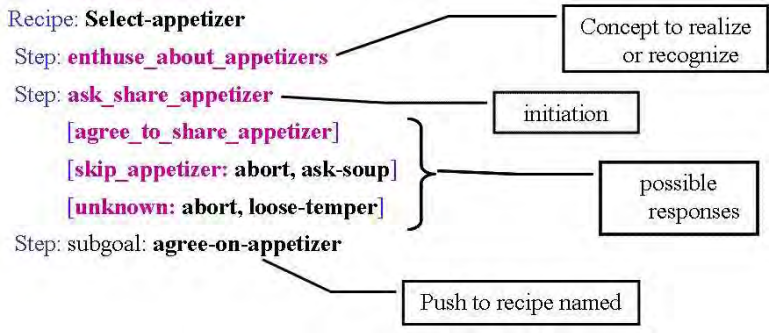

Figure 4: A basic script

When writing a recipe for a topic the author creates one or more steps. A primitive step is an initiation that can be optionally paired with a set of anticipated responses. The first step in Figure 4 shows a step with only an initiation and no 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. The main difference between initiation and response specifications is that more than one response concept can be specified as with the second step in Figure 4, the items in square brackets are different anticipated responses.

The initiation and response specifications can be used by the system for both interpretation and generation. For example, in Figure 4, the student may have expressed the initiation concept ask_share_appetizer

and when the system recognizes this initiation concept then it will randomly select one of the anticipated response recipes to express as part of its dialogue turn.

Steps that are pointers to sub-recipes enable hierarchically organized dialogue. A recipe can embed another recipe by referring to the goal name of that recipe in a step as with the last step in Figure 4. The effect is that there is a push to the embedded recipe and when the embedded recipe completes, control returns to the parent.

Responses can be authored to include follow-up actions so that flawed or vague responses can be addressed as needed. Follow-up actions, such as **abort** and **ask-soup** in the second response in step two, are expressed as an ordered list of recipes for subdialogues that are to be invoked in response when the understanding module determines that the student's response best fits the concept listed. The recipe, *abort*, is a reserved recipe name. This response action indicates that the parent recipe should be aborted. An *abort* can appear anywhere in the list of response actions but everything in the list will be executed before the abort of the parent recipe happens.

Finally a reserved concept identifier of *unknown* should always be included and an appropriate response recipe specified as with the last response concept in the second step of Figure 4. The *unknown* choice is followed only if none of the expected response concepts matched. If student initiative is enabled for the system, then there is the additional restriction that no show of initiative (see definition below in section on student initiative) is detected before the *unknown* choice is selected.

An initiation can also result in an expectation for multiple response concepts to provide a complete response. Responses that require these complex answers allow the dialogue manager to credit the student with fulfilling part of a discourse obligation and then pursue only the unfulfilled aspects of the obligation. Assuming a concept such as *do-you-want-soup-and-salad* includes phrases that means "Do you want soup and salad?" and excludes phrases that mean "Do you want soup or salad?", then it sets up a discourse obligation to answer about both soup and salad. Although it is possible to set up specialized concepts for a single concept that combine answers about both soup and salad, this is dispreferred because if the student answers only about soup or only about salad, then the system cannot count any discourse obligation as having been fulfilled.

To signal that a multiple part response is anticipated, a list of the necessary and sufficient response concepts must be specified for a step. Even if a multiple part response is not specified, it is possible to recognize multiple concept matches and perform whatever actions are indicated for each match. But only the best match for a span of a response is used so multiple response matching applies only to different spans of a student response. The follow-up actions for a non-match are similar to those for a match (as discussed earlier) but only non-match follow-up actions apply to multi-part answers and specify the actions to take when a **necessary and sufficient** response part is missing.

The reserved concept *unknown* is selected only if none of the necessary and sufficient answer concepts matched. The *unknown* response is matched only if none of the other responses matched. If the system is configured to allow student initiative then *unknown* is selected only when all matches fail and no show of initiative is detected.

If alternative ways of expressing a concept are defined for an initiation or response then an available policy for choosing between them is to check which category of utterance was last selected and if answered appropriately to choose the next hardest version of the utterance. If it was answered inappropriately then it chooses the next easiest version of the utterance. The type of each alternative utterance is specified by the difficulty level attribute in its concept entry (as previously discussed).

An author can also write one or more recipes for each dialogue topic or subtopic. Each recipe provides alternative ways of covering that topic. Adding an optional level of difficulty to each alternative provides an additional context check that helps the system choose between multiple versions of recipes with the same goal name similarly to when there are alternative ways to express a concept. If no level of difficulty is supplied, the default behavior is to choose the least recently used recipe for the goal when the system is to initiate discussion of the topic.

A recipe step can be assigned an optional semantic identifier. The intent is that all steps with similar meaning would be assigned the same identifier. This allows the dialogue system to track this information across dialogue turns and react to repeats of the identifier according to the policy selected by the author during configuration of the system.

If a recipe step is marked optional then one of the available policies is to skip the step if the semantic identifier on the initiation appears in the recent dialogue history or to skip the step if the semantic identifier appears less recently in the dialogue history but was answered appropriately or is inappropriate to repeat (e.g. was said by the dialogue partner). For example, if a recipe R1 has been invoked recently and is to be invoked again then any semantic identifiers in R1 will be in the dialogue history and any

optional steps with one of those identifiers will be skipped on the current invocation of the recipe.

With optional steps, an author can create a recipe in which only the most context-relevant steps are covered. With the capability for mixed-initiative for topics, recipes that make use of looping and optional steps can result in a dialogue in which the steps of the recipe are not always performed sequentially and can instead be executed in an order initiated by the student.

Authors can also assign a semantic identifier to a recipe as well as to recipe steps. So if recipe S1 has a semantic identifier and S1 is embedded as an optional step in P1, S1 will be skipped if its identifier appears in the discourse history and the other conditions indicated by the selected default policy hold.

The scripting language also makes use of semantic identifiers as termination conditions to implement looping of an entire recipe. A recipe that is to loop is specified by including the goal attribute *retry-until-cover* which encodes the termination condition. All semantic identifiers included in the termination condition must appear in the dialogue history before the loop will terminate.

Initiative

Currently the system provides a very crude means of responding to student initiative in order to explore the potential of chat dialogues for language learning applications. We define student initiative as any part of a student's response that matches a concept outside of the current step that was not previously recently matched according to the dialogue history (i.e. what was said earlier may have been repeated for increased coherency instead of being a show of initiative). Currently the system will either always accept or reject an initiative depending on how the system is configured. No overriding of the set policy of either always accept or always reject is currently possible. First we will describe what happens when the policy is always accept or always reject. Then we will discuss potential problems we anticipate with not allowing overrides of the policy and possible solutions we have discussed so far.

If the default policy in the configuration is to accept any initiative, the current version of system will simply accept the initiative and start responding to it. But if some of the matched concepts are relevant to the current topic, the system will pursue those first and then will respond to the initiative. When the initiative topic is completed the system will attempt to resume whatever was interrupted. The transition from a completed initiative back to what was interrupted has the potential to lead to incoherencies in the current version of TuTalk.

A problem with an accept all or none policy on student initiative is that it may sometimes be more appropriate to override the default policy. In the case of always accepting student initiative, it may sometimes be more appropriate to ignore or postpone the initiative depending on the relative importance of the current step and the initiative. In the case of always ignoring initiative it may sometimes be more appropriate to accept or postpone the initiative. Again it would depend on the relative importance of the current step and the initiative.

In addition to knowing when to override the default, it is tricky to properly implement postponing or accepting initiative. To properly postpone, perhaps the system should offer an acknowledgment of the initiative that communicates it has been understood and will be entertained once the current topic is completed such as "first answer my question and then we'll talk about <topic>". Then it should put the postponed initiative as a goal on the dialogue manager's stack and then retry the current step. Deciding where to put the postponed initiative on the stack is not straightforward either.

Simply interrupting the current goal to accept an initiative is also a potential problem. The proper acceptance of an initiative means deciding whether to retry the current goal or some higher level goal on the dialogue manager stack after pursuing the initiative or instead completely pruning the current goal or some higher level goal from the stack.

To consider the relative importance of the initiative we would need to allow authors to specify a separate relative importance for semantic identifiers or add an importance attribute to concept definitions If the global policy is to ignore initiative then if the relative importance of the two semantic tags or concepts for the step initiation and the initiative match indicates that the initiative is more important than the current step then the global policy could be overridden and the initiative taken up by the system and the current goal either aborted or retried at a later time. The inverse would apply if the policy is to accept all initiatives. If relative importance cannot be decided then the global policy would continue to be followed.

FINAL REMARKS

We have described the TuTalk dialogue system construction tool. It provides a plug-and-play type of dialogue system for learning applications that facilitates integration with new modules and experimen-

tation with different NLP modules, configuration options that effect the behavior of the plug-and-play NLP modules so that it can be flexibly fine-tuned for a large number of different experiments, and an authoring language for setting up the domain knowledge and resources needed by the NLP modules.

By providing handling of student topic initiative, the tool has the ability to support peer as well as tutorial style dialogues. One experiment that is being considered will make use of student topic initiative to test chat type dialogues like those sometimes used in language learning classes. Another experiment will use TuTalk to model a peer during collaborative problem solving.

ACKNOWLEDGMENTS

This work was funded by the National Science Foundation through award number SBE-0354420 to the Pittsburgh Science of Learning Center at Carnegie Mellon University and the University of Pittsburgh.

REFERENCES

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

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

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

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

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

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

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

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

[McTear, 2002] McTear, M. (2002). Spoken dialogue technology: enabling the conversational user interface. *ACM Computing Surveys*, 34(1):90–169.

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

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

[Traum et al., 1999] Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., Matheson, C., and Poesio, M. (1999). A model of dialogue moves and information state revision. Technical report, Trindi Project Deliverable D2.1.

[VanLehn et al., 2002] VanLehn, K., Jordan, P., Rosé, C., Bhembe, D., Böttner, M., Gaydos, A., Makatchev, M., Pappuswamy, U., Ringenberg, M., Roque, A., Siler, S., and Srivastava, R. (2002). 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.

[Young et al., 1994] Young, R. M., Pollack, M. E., and Moore, J. D. (1994). Decomposition and causality in partial order planning. In *Second International Conference on Artificial Intelligence and Planning Systems*. Also Technical Report 94-1, Intelligent Systems Program, University of Pittsburgh.