

Responding to and Recovering from Mistakes during Collaboration

Andrew Garland, Neal Lesh, and Charles Rich

Mitsubishi Electric Research Laboratories

201 Broadway, Cambridge MA 02139

{garland,lesh,rich}@merl.com

Abstract

Successful collaboration requires the participants to maintain mutual understanding of their shared goals and plans. This paper, unlike prior work, addresses how to maintain these mutual beliefs when a user performs a *non-contributor*, i.e., a mistake or an interruption. We present methods to infer the intentions underlying mistakes, to repair shared plans by adding recovery goals, and to generate utterances that maintain mutual understanding about both non-contributors and recoveries. By explicitly representing recovery goals, our system is able to discuss and achieve recoveries in the same manner as any other subgoal.

1 Introduction

There has been a great deal of research toward making intelligent agents collaborate with human users [5, 15, 14]. Successful collaboration requires the participants to maintain mutual understanding of their shared goals and the actions intended to achieve them, through both verbal communication and performing actions. Previous work has focused on maintaining mutual understanding when the participants execute “correct” actions, i.e. those which contribute to a shared goal.

This paper extends previous work by addressing the case when a user performs a *non-contributor*, i.e., either a mistake or an interruption (an action that contributes to a new goal). We present novel methods and discuss open issues on several relevant topics: how to infer the intentions underlying mistakes, when and how to repair the shared plan by adding explicit recovery goals, and how to generate responses that maintain mutual understanding about both non-contributors and recoveries.

While the ideas we discuss apply to collaboration in general, the clearest motivation and examples occur in the context of teaching. For example:

User presses the engage button.
◇Agent says “Whoops, it was too soon to press the engage button.”
◆Agent says “Let’s recover from pressing the engage button.”
◇Agent says “First, let’s set the engine speed to zero”
⋮

◆Agent says “We’ve recovered from pressing the engage button.”

◇Agent says “Let’s return to starting the engine”

Later, in a similar context, the interaction could be:

User presses the engage button.

◆Agent says “Whoops, you’ve made this mistake before.”

◇Agent says “You take it from here.”

User says “What next?”

◆Agent says “Let’s recover from pressing the engage button.”

Previously we have reported on an intelligent tutor [16, 13] that could generate the utterances marked with a ◇, which include negative feedback and instruction on the actions needed to recover from a mistake. However, it did not distinguish between actions typically needed to achieve the current goal and ones that are part of recoveries, and could not produce the comments marked with a ◆.

This highlights a criterion for maintaining mutual understanding during recovery: an agent must be able to indicate when recovery is needed, and which actions are part of the recovery. This matches what people do in many domains, e.g., “here’s what you do if you accidentally engage the engine prematurely ...”.

A second criterion is to reuse as much of the agents general capabilities as possible. For example, the same mechanisms that the agent would use to discuss and decide between alternate ways to achieve some domain goal should be used to discuss and decide between alternate ways to recover from a mistake.

Our approach to meeting these two criteria is to explicitly represent recovery goals and add them to shared plans during repair. Since recoveries are represented like any other subgoal, our generic mechanisms for communication and planning operate on them correctly. It is also clear when actions are part of recoveries since they are subplans of recovery goals. Furthermore, this approach allows a tutor to intentionally commit mistakes in order to teach how to recover from them.

We developed these techniques in order to extend COLLAGEN [14], an application-independent collaboration manager based on SharedPlan theory [6, 11], which we used to develop our intelligent tutor. We will, however, attempt to present our techniques as generally as possible.

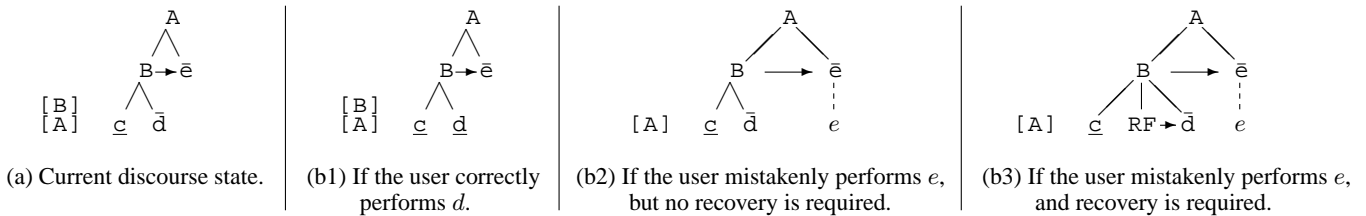


Figure 1: Examples of interpretation and repair.

2 Background

Collaboration is a process in which two or more participants coordinate their actions in order to achieve shared goals. In this paper, we focus exclusively on collaborations involving only two participants. Tutoring is a kind of collaboration in which one participant (the tutor) has greater expertise and initiative, and the primary shared goal is to increase the student’s expertise. Assistance is another kind of collaboration in which the initiative may vary and the primary shared goal is to successfully accomplish some domain task.

A *task model* specifies domain tasks and possible ways to accomplish them. Specifically, a task model includes primitive actions, non-primitive actions, and recipes. Primitive actions include both “physical” actions, such as pressing a button, and utterances. Non-primitive actions are achieved by decomposing them into other actions. A recipe is a method for decomposing a non-primitive, possibly imposing precedence relations among the steps, as well as other logical relations among their parameters.¹ In general, there are one or more recipes for each non-primitive action type, which may be chosen based on the recipe’s applicability conditions.

Discourse is a technical term for an extended communication between two or more participants in a shared context, such as a collaboration. Previous work [14] has discussed at length the crucial role that collaborative discourse theory [6, 11] has played in the development and architecture of COLLAGEN. Our discourse state representation consists of a stack of goals, called the *focus stack*, and a *plan tree* for each goal on the stack. The top goal on the focus stack is the current task or subtask. The plan tree associated with each goal gives its hierarchical decomposition, if any, into partially ordered sets of subtasks with constraints.

Figure 1(a) shows a “snapshot” of a discourse state. Uppercase letters, such as A, denote non-primitive actions. Lowercase letters denote primitive actions; those that are underlined, such as \underline{c} , have been done and those with a bar on top, such as \bar{e} , need to be done. In Figure 1(a), the focus stack is on the left, and shows that B is the current task. To the right is the plan tree, which shows the decomposition of both A and B. The arrow between B and \bar{e} indicates a precedence link, i.e. B must be achieved before \bar{e} can be done.

After the user or agent performs an action, *discourse interpretation* updates the discourse state. The basic job of discourse interpretation is to explain how the most recently performed action, called the occurrence, *contributes* to the collaboration. A physical action typically contributes by match-

ing one of the steps in a plan. An utterance typically contributes by referring to one of the steps in a plan. Updating the discourse state typically involves some combination of extending the plan tree and popping and/or pushing the focus stack.

During interpretation of an occurrence o , an embedded plan recognizer [9] considers ways to extend a current plan that respect the constraints in the task model and include performing o . We refer to such an extension as an *explanation* for o . The agent only considers explanations that respect all the ordering and precondition constraints in the task model. Further, the agent does not generate explanations in which o is a descendant of an action that is already known to be achieved.

Figure 1(b1) demonstrates how discourse interpretation normally works. In this case, the user has just performed the primitive action d . Since the current plan contains a \bar{d} step, a correct explanation exists; after the discourse state is updated, the step is denoted as \underline{d} since it no longer needs to be done. d is added as a subplan of \underline{d} . (Figure 1 does not show the subplans of \underline{c} and \underline{d} to keep the pictures smaller.)

3 Interpreting non-contributors

We now define a *non-contributor* as an occurrence for which the plan recognition algorithm described above finds no explanation. There are two possible interpretations for non-contributors. As discussed in [10], it is always possible that the user has interrupted the current task in order to work on some new task. In addition, some occurrences just nearly miss contributing to a current shared goal, so it is reasonable to interpret them as mistakes.

When both interpretations are possible, the system must decide which is the more likely. We have experimented with different methods for making this decision, but generally prefer to assume that an occurrence is a mistake rather than an interruption. This enables the agent to provide more specific feedback to non-contributors, as will be shown in Section 6.

Our method for determining if a non-contributor can be interpreted as a mistake is to search for “near miss” explanations. Near-misses are found by re-running the plan recognizer while relaxing ordering, precondition, and postcondition constraints. Another type of near-miss explanation can be found by relaxing constraints between parameter values.² This explains actions that would have been correct if the occurrence had had different parameters. For example, if the plan called for the engine speed to be set to 12 but the person set it to 10, we would interpret this as a mistake by the user.

¹These relations are also referred to as constraints.

²This is only partially implemented at this time.

Figure 1(b2) demonstrates discourse interpretation after a “harmless” mistake has occurred. In this case, the user performs e immediately after c ; this is a mistake because B is not yet achieved. As a result, e is added as a *non-contributing subplan* of \bar{e} , which is indicated by the dashed link between them. This mistake is considered harmless because of domain knowledge, as will be explained in the next section. \bar{e} still needs to be done, but \bar{d} still must be done first.

4 Repairing the shared plan

Regardless of why a non-contributor was executed or whether it was a mistake, a shared plan may need to be repaired. A key aspect of the repair algorithm is that the only changes made are the addition of one or more recovery nodes and precedence links involving those recoveries. This strategy allows recovery to be explicitly discussed (or taught) without requiring any extension to the agent’s general abilities. As will be discussed in Section 5, there are some situations that the techniques described here do not address.

Figure 2 contains pseudo-code for our plan repair algorithm, called REPAIR. It is not the case that every single node in the plan tree is repaired after every single non-contributor. Determining which nodes need repair is based on general principles applied to specific domain knowledge. The principles identify those nodes in the plan tree that are *vulnerable* to the occurrence; whether a vulnerable plan actually needs to be repaired depends on domain-specific knowledge.

In general, the plans that are vulnerable are the most-deeply-nested, started plans that have not yet been achieved (see function ISVULNERABLE in Figure 2). If a plan has not been started yet, then the occurrence cannot interfere with its decomposition. If a node of type *Foo* has been achieved and the occurrence interferes with its effects, then the node’s parent may need to be repaired (possibly by retrying *Foo*) but the node itself does not.

Figure 1(b3) shows an example of plan repair when e is done too early, and this interferes with the current shared goal of B (B is vulnerable, and A is not). In this case, the only correct action to work on is the newly added recovery node (labeled RF, which is an abbreviation for *RecoverFrom*(B, e)). This plan tree does not show a decomposition for RF; one would be chosen by the same mechanisms by which a recipe is chosen for any non-primitive.

The reason that a plan is not vulnerable if it contains a subplan that is started and not achieved is because of the encapsulation of abstract acts. Any valid decomposition of a subplan is acceptable as far as the plan is concerned as long as the constraints on its decomposition are not violated. We are positing that the semantics of *RecoverFrom* respects this principle of encapsulation; thus, our repair algorithm transforms one valid decomposition into another valid decomposition. For example, in Figure 1(b3), the plan for A is not vulnerable because any needed repair will be part of the new decomposition for B.

Domain knowledge is necessary to determine if a vulnerable plan actually needs to be repaired because of a non-contributing occurrence. If the task model includes causal

```

REPAIR ( $o$ )  $\equiv$ 
  forall shared plan trees  $T$ 
    REPAIRPLAN(ROOT( $T$ ),  $o$ )

REPAIRPLAN ( $P, o$ )  $\equiv$ 
  if ISVULNERABLE( $P$ )
    if REQUIRESRECOVERY( $P, o$ )
      ADDRECOVERY( $P, o$ )
    else forall  $s$  in SUBPLANS( $P$ )
      if  $\neg$  ISACHIEVED( $s$ )
        REPAIRPLAN( $s, o$ )

ISVULNERABLE ( $P$ )  $\equiv$ 
  if ISACHIEVED( $P$ ) or  $\neg$  ISSTARTED( $P$ )
    return false
  forall  $s$  in SUBPLANS( $P$ )
    if ISVULNERABLE( $s$ )
      return false
  return true

ADDRECOVERY ( $P, o$ )  $\equiv$ 
   $s \leftarrow$  new RecoverFrom( $P, o$ )
  add Precedes( $s, \text{SUBPLANS}(P)$ ) to the constraints of  $P$ 
  add  $s$  as a subplan of  $P$ 

```

Figure 2: Pseudo-code to repair a shared plan tree after an occurrence o .

links, then REQUIRESRECOVERY can be computed from those links. Otherwise, values for REQUIRESRECOVERY must be explicitly specified. While specifying those values might increase the modeling effort for a given domain, default rules usually suffice. An optimistic domain modeler would assume that recovery is not needed by default, and override this in very specific cases; a pessimistic model would assume that all physical actions require recovery. In Figure 1(b2), REQUIRESRECOVERY(B, e) is false so no *RecoverFrom* is added to the shared plan; in Figure 1(b3), REQUIRESRECOVERY(B, e) is true.

An important special case for recovery is when some or all of the domain actions are *reversible*, as in direct manipulation interfaces. When an action x is reversible, then there is a sequence of one or more actions (possibly just the action itself, e.g., for a toggle button) that form a decomposition for *RecoverFrom*(G, x) for any goal G . While this decomposition will always be applicable, it may sometimes be “too strong” — for example, if a non-contributor had ten effects, perhaps nine of them are beneficial and only one is detrimental with the regard to the current plan.

The last step of repair is to add a new *RecoverFrom* subplan and enough precedence relationships to make sure that recovery is done next. The pseudo-code in ADDRECOVERY does this very crudely; our implementation only adds the minimal number of precedence links required.

Note that the techniques presented here will work even when several non-contributing actions are made in a row. This method adds a *RecoverFrom* subplan for each and precedence links between them such that the non-contributors will be recovered from in reverse order.

5 Open issues

There are several important issues regarding repairing the shared plan that are not handled by the algorithms presented in this paper.

First-principles planning REPAIR requires that the domain model will contain decomposition rules for *RecoverFrom*, and each rule will be associated with applicability checks to make sure it is only used when appropriate. This requirement is perfectly reasonable for COLLAGEN or other systems that “plan” by recipe selection. However, this seems an unnecessary modeling burden in systems that include a first-principles planner (and must, therefore, already include a full causal model).

It might be possible to avoid this modeling burden; here is the gist of one potential approach, which relies on modifying the first-principles planner slightly. The modified planner will infer a decomposition of *RecoverFrom* by comparing *P* with a “augmented” copy of *P*. The augmented copy of *P* is produced by generating a plan for the current goals using the planner, with the restriction that the planner can only add subplans to *P* or mark subplans as no longer needed, i.e. the planner cannot retract any of the decompositions that are already in *P*. Also, *P* can only be augmented if it is vulnerable. Then, comparing AUGMENTED(*P*) to *P* will identify the actions in AUGMENTED(*P*) that should be grouped under the recovery node. The final step would be to copy all of the markings of AUGMENTED(*P*) to *P*, and copy the constraints between the recovery actions to the *RecoverFrom* node.

Non-local repairs The repair algorithm presented in this paper assumes that all necessary recovery should be done as one atomic unit and, when recovery is over, that the rest of the plan can proceed normally. *Non-local repairs* are necessary when this assumption does not hold. Some non-local repairs can be accommodated by more complicated plan structures that force actions to be interleaved to achieve multiple subgoals (namely the recovery and the original goal). In more dramatic cases, the agent will need to indicate that the non-contributor has significantly effected the plan, and so it can no longer be executed as was originally planned, and then proceed with the new plan (e.g., you break the jack while changing the tire, so now have to call for help).

Imperfect knowledge Repair might be needed due to imperfect action (or recipe) descriptions or other source of unanticipated changes to the state of the world. In such situations, recoveries may be sufficient or repair could require changing the decomposition for a non-primitive. In addition, the system could misidentify a contributing action as a non-contributor if the underlying domain model is in error. In general, imperfect knowledge would require very different techniques than those presented here.

Ambiguity Our techniques can generate multiple near-miss explanations. Dealing with ambiguity is a general problem in that there can also be multiple correct explanations for an occurrence. The general strategies for dealing with ambiguity include picking an explanation based on some preference

rules, asking a clarification question, and waiting for future actions to disambiguate. These can all be applied to near-miss explanations as well, though some may be less attractive since the intention inference is fundamentally more speculative. The techniques presented in this paper will not be able to repair plans or guide recoveries until the ambiguity is resolved.

“Bumping” It may seem strange that our algorithm can add *RecoverFrom* subgoals to two vulnerable subplans. It would seem that it should be sufficient to recover from the non-contributor once. However, this is a general problem that potentially occurs whenever the same subgoal appears twice in a plan. It may well be the case (but is not always the case) that it only needs to be achieved once. In the planning literature, this problem is referred to as “bumping” because one subgoal is achieved by using the actions from another. COLLAGEN currently does not have a systematized method for handling bumping, but such a facility would allow for two *RecoverFrom* subgoals to be achieved simultaneously.

Goal maintenance Another set of issues arises from the fact that recovering from a non-contributor typically requires undoing some or all of its effects. However, it is possible that the user caused those effects intentionally. For example, if the user opens a window on her computer screen to check on her stocks, a “helpful” assistant agent might immediately close that window if it obstructs the task that was being collaboratively achieved. Again, this is really part of a larger issue having to do with understanding which goals need to be maintained (and for how long) and which ones do not.

6 Maintaining shared beliefs

It is important for the utterances and actions of the agent to maintain shared beliefs about non-contributors and recoveries. In our system, reacting to non-contributors required some new methods. However, the actions of the agent during recovery are produced by the same general mechanisms for producing behavior when achieving any goal.

The agent now “watches” for non-contributors, and responds with an utterance that is generated from the near-miss explanation. For example, if the user performs a step that is ordered after an action that has not yet been done, then the agent says “Whoops, you can’t do that yet.” If the user performs an action that they have already performed (and does not need to be repeated), then the agent says “Whoops, you already did that.” The agent has similar utterances for actions with false preconditions or postconditions that were already true before they were executed.

Figures 3 and 4 contain sample execution traces with our agent demonstrating the ways in which our agent maintains mutual understanding about non-contributors and recoveries. These indented traces are automatically produced by the discourse interpretation algorithm, and provide useful summaries of the interaction. In these figures, the lines in italics describe non-primitive acts, and the other lines describe primitive acts.

Achieving A.

⋮
Agent performs c.
User performing d.
Agent says "Please perform d."
Done agent performing c.
User inappropriately performs c.
Agent says "Whoops, c was already done, but that was a harmless mistake."
Agent says "Please perform d."
Stopped interruption.
User performs x.
Agent says "Whoops, your performing x was not relevant to our task."
Done recovering from performing x.
Done agent performing UndoX.
Agent says "I'm going to perform UndoX."
User says "Ok."
Agent performs UndoX.
Agent says "Please perform d."

Figure 3: Sample behavior by an intelligent assistant.

Showing how to achieve A.

⋮
Showing how to perform d.
Agent says "d is needed because..."
Agent says "Now, perform d (click the button at which I am pointing)."
Blocked performing e.
User inappropriately performs e.
Agent says "Whoops, your performing e was premature."
Done showing how to recover from performing e.
Agent says "Recovering from prematurely performing e is needed because..."
Agent says "Now, we should recover from performing e."
User says "Ok."
Done showing how to perform UndoE.
Agent says "First, you should perform UndoE."
User performs UndoE.
Agent says "Good job."
Agent says "You succeeded in recovering from performing e."
Blocked performing e.
User performs e.
Agent says "Whoops, you've made this mistake before."
Done showing how to recover from performing e.
Agent says "You take it from here (say 'what next?' for additional help)."
User says "What next?"
Agent says "Now, we should recover from performing e."
Done showing how to perform UndoE.
User performs UndoE.
Agent says "Great."
Agent says "Please perform d."

Figure 4: Sample behavior by a tutoring agent.

In Figure 3, the agent is acting as an assistant; in Figure 4, the agent is acting as a tutor. In both cases, the discourse state is being updated in the same way. However, since the basic response mechanisms for assistants and tutors differ, the behavior of the agent differs in the two figures. For example, the assistant is willing to proactively start doing the actions needed to recover, whereas the tutor treats recovery as another opportunity to teach. Another example is that the tutor generically points out when any non-primitive action is

accomplished, and thus says "You succeeded in recovering from performing e."

The utterance "Whoops, you've made this mistake before," is generated in a generic way, by taking advantage of a *user model*. By consulting the user model, the agent can identify when mistakes are repeats, can identify when the user knows how to accomplish tasks ("You take it from here"), and can suppress repeating explanations (such as explaining why recovering from prematurely performing e is needed).

Note that it both difficult and risky to try to identify the misconception in the user's head that led them to execute an incorrect action. For example, if the user executed some action c that should have been executed after b , they may not have known about the ordering constraint or they may have mistakenly thought that b didn't need to be performed in this case because its postcondition was fortuitously true. Our agent's response, "Whoops, you can't do that yet", is accurate and hopefully helpful in either case.

As another example, suppose a user executes action d that she thinks is necessary to achieve the current goal A . If d never contributes to A , our agent will say "Whoops, d was not relevant to our task." But if d 's postcondition was fortuitously true, our agent will say "Whoops, you didn't need to do that". We believe these different types of feedback help the user correct their misconception and learn the tasks, without requiring the agent to know exactly which misconception the user has.

7 Related research and conclusion

There has been an impressive variety of work on algorithms and heuristics for repairing plans (e.g., [7, 8, 12]). Some of this work addresses repairs that we have not considered, such as replacing an action that just failed with a different action to achieve the same goal [1]. These methods typically operate on a nearly-correct plan and attempt to minimally add or remove actions in order to make it correct. This work is complementary to our efforts in that we present techniques for inferring the intentions of mistakes, and techniques for how to incorporate recovery actions into a plan, not on how to generate the necessary recovery actions.

The STEVE tutoring system [15] has similar behavior to our system, but neither represents or discusses error recoveries. There has also been a great deal of work on tutoring systems in math and physics (e.g., [4]) in which the tutoring system diagnoses and provides feedback on the errors made by the students. These domains are extremely different than the procedural tasks we have been working on. For example, no recovery is required, *per se*, when a mistake is made while solving a math problem.

Probabilistic plan recognition (e.g., [3, 2]) can be robust to user mistakes, but does not try to diagnose or recovery from them as do the techniques in this paper. This paper presumes that decomposition knowledge would be applied without specifying how it could be acquired, which might be through learning or a knowledge-acquisition tool [17].

In conclusion, this work extends prior research so that collaborators can maintain mutual beliefs when a user performs a non-contributor, be it a mistake or an interruption. We presented methods to infer the intentions of mistakes, to repair shared plans, and to generate utterances about both non-contributors and recoveries. Our methods explicitly represent recovery goals, so our generic mechanisms for interpretation and planning operate on recoveries as they would on any goal.

References

- [1] R. Alterman. Adaptive planning. *Cognitive Science*, 12:393–421, 1988.
- [2] M. Bauer, S. Biundo, D. Dengler, J. Kohler, and G. Paul. PHI — A Logic-based Tool For Intelligent Help Systems. In *Proc. 13th Int. Joint Conf. AI*, pages 460–466. 1993.
- [3] E. Charniak and R. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.
- [4] C. Conati, A. Gertner, K. VanLehn, and M. Druzdzel. On-line student modeling for coached problem solving using Bayesian networks. In *Proc. 6th Int. Conf. on User Modelling*, pages 231–242, Sardinia, Italy, 1997.
- [5] G. Ferguson and J. Allen. TRIPS: An integrated Intelligent Problem-Solving Assistant. In *Proc. 15th Nat. Conf. AI*, pages 567–572, 1998.
- [6] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.
- [7] K. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.
- [8] Steven Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *J. of Artificial Intelligence Research*, 2:319–360, January 1995.
- [9] N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proc. 7th Int. Conf. on User Modelling*, pages 23–32, Banff, Canada, June 1999.
- [10] N. Lesh, C. Rich, and C. Sidner. Collaborating with focused and unfocused users under imperfect communication. In *Proc. 9th Int. Conf. on User Modelling*, pages 64–73, Sonthofen, Germany, July 2001.
- [11] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4):525–572, December 1998.
- [12] Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, 76:427–454, 1995.
- [13] C. Rich, N. Lesh, J. Rickel, and A. Garland. A plug-in architecture for generating collaborative agent responses. In *Proc. 1st Int. J. Conf. on Autonomous Agents and Multiagent Systems*, pages 782–789, Bologna, Italy, July 2002.
- [14] C. Rich, C. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, Winter 2001. Special Issue on Intelligent User Interfaces.
- [15] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343–382, 1999.
- [16] J. Rickel, N. Lesh, C. Rich, C. Sidner, and A. Gertner. Using a model of collaborative dialogue to teach procedural tasks. In *Proc. 10th Int. Conf. on Artificial Intelligence in Education*, pages 592–594, San Antonio, TX, May 2001.
- [17] G. Tecuci, M. Boicu, K. Wright, S. Lee, D. Marcu, and M. Bowman. An integrated shell and methodology for rapid development of knowledge-based agents. In *Proc. 16th Nat. Conf. AI*, pages 250–257, 1999.